НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Система моніторингу успішності в шкільних закладах»

Виконав (-ла):
студент (-ка) IV курсу, групи ІТ-51
Дмитрук Максим Володимирович _____

Керівник:
ст. викладач Яланецький В.А. _____

Рецензент:
Директор ТОВ АДРАБА ТЕХНОЛОДЖІС Україна Лімітед
Т.О. Шевчук _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О.І. Ролік

«___»_____2019 р.

ЗАВДАННЯ

на дипломний проект студенту

Дмитруку Максиму Володимировичу

1.Тема проекту «Система моніторингу успішності в шкільних закладах», керівник проекту Яланецький Валерій Анатолійович, старший викладач, затверджені наказом по університету від «___»_____ 2019 р. №____’_

2. Термін подання студентом проекту

_____

3. Вихідні дані до проекту кількість користувачів від однієї школи – 1750(550 учнів, 1100 батьків та 100 вчителів), взаємодія користувачів з системою через клієнтські додатки.

4. Зміст пояснювальної записки огляд існуючих систем моніторингу успішності в шкільних закладах, формування базових вимог до системи, опис обраного набору технологій, опис принципу роботи системи, опис процесу розробки системи, керівництво використання системи, процесу її інтегрування та додатки.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо) блок-схема алгоритму обробки запитів за допомогою реалізацій інтерфейсів ICommandBus та IQueryBus, діагра-

ма зв'язків між сутностями, Use-case діаграма системи, Діаграма послідовності авторизованого запиту розкладу

6. Дата видачі завдання 15 березня 2019

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Вибір теми дипломного проекту	10.02.2019 – 08.03.2019 рр.	
2	Пошук та аналіз матеріалів відповідно до обраної теми	20.03.2019 – 01.04.2019 рр.	
3	Аналіз поставленої задачі, формування принципу роботи системи та технічних і функціональних вимог	03.04.2019 –10.04.2019 рр.	
4	Проектування архітектури системи	11.04.2019 – 17.04.2019 рр.	
5	Вибір набору технологій для реалізації системи	18.04.2019 – 25.04.2019 рр.	
6	Розробка серверної частини	26.04.2019 – 01.05.2019 рр.	
7	Розробка клієнтської частини	01.05.2019 – 17.05.2019 рр.	
8	Написання та оформлення пояснювальної записки до дипломного проекту	15.05.2019 – 04.06.2019 рр.	
9	Захист дипломного проекту	18.06.2019 рр.	

Студент М.В. Дмитрук

Керівник проекту В. А. Яланецький

АНОТАЦІЯ

Дипломний проект містить пояснювальну записку обсягом 65 аркушів. Пояснювальна записка складається з 5 розділів та містить 20 рисунків, 5 додатків і 11 посилань на джерела.

Перелік ключових слів: автоматизація процесів, інтеграція з системою, моніторинг успішності, навчальний заклад, система моніторингу.

Метою системи є економія часу та підвищення ефективності роботи учасників навчального процесу шляхом автоматизації дій, які вони виконують. Поставлену задачу виконано за допомогою реалізації системи моніторингу успішності в шкільних закладах.

Система складається з Web-застосунків, кожен з яких має клієнт-серверну архітектуру. Серверна частина написана на мові програмування C#. Вона являє собою WebAPI застосунок на платформі ASP.NET Core. В якості сховища даних використано СУБД MSSQL Server. Клієнтська частина системи реалізована з використанням JavaScript фреймворку React. Система може бути розгорнута на всіх найбільш популярних сучасних операційних системах. Також система може бути використана для створення власних клієнтських додатків на будь-яких платформах.

ANNOTATION

The diploma project contains an explanatory note in the volume of 65 sheets. The explanatory note consists of 5 chapters and has 20 drawings, 5 supplements and 11 references to sources.

Keywords: automation of processes, integration with the system, monitoring of performance, educational institution, monitoring system.

The purpose of the system is to save time and improve performance of studying process participants by automating the actions they perform. The task is done through the implementation of system for monitoring the performance of school establishments.

System consists of web-applications, each of them implements client-server architecture. Server part is written using C# programming language. It is WebAPI application on ASP.NET Core platform. Database management system MSSQL Server is used to store data. Client part of the system is implemented with JavaScript framework React. System could be deployed to any of the most popular and modern operation system. System can be used to create own client applications for any platform.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІТ51.070БАК.001 ТП	Відомість технічного проекту	1	
3	А4	ІТ51.070БАК.002 ПЗ	Пояснювальна записка	64	
4	А4	ІТ51.070БАК.003 Д1	Додаток А. Блок-схема алгоритму обробки запитів за допомогою реалізацій інтерфейсів ICommandBus та IQueryBus	1	
5	А4	ІТ51.070БАК.004 Д2	Додаток Б. Діаграма зв'язків між сутностями	1	
6	А4	ІТ51.070БАК.005 Д3	Додаток В. Use-case діаграма системи	1	
7	А4	ІТ51.070БАК.006 Д4	Додаток Г. Діаграма послідовності авторизованого запиту розкладу	1	
8	А4	ІТ51.070БАК.007 Д5	Додаток Д. Вихідний код програми	48	

	ПІБ	Підп.	Дата			
Розробн.	Дмитрук М.В.				Лист	Листів
Керівн.	Яланецький В.А.				1	1
Н/контр.						
Зав.каф.	Ролік О.І.					

Пояснювальна записка
до дипломного проекту
на тему: «Система моніторингу успішності в шкільних закладах»

Київ – 2019 рік

.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 5

ВСТУП 6

1. АНАЛІЗ СИСТЕМ МОНІТОРИНГУ УСПІШНОСТІ В ШКІЛЬНИХ ЗАКЛАДАХ 8

1.1 Актуальність системи для різних типів навчальних закладів ... 8

1.2 Причини використання подібних систем 9

1.3 Огляд існуючих рішень для моніторингу успішності в шкільних закладах 11

1.3.1 Шкільна освітня мережа «Щоденник.ua» 11

1.3.2 Система «Електронний журнал» 12

1.3.3 Освітній портал «Класна оцінка» 13

1.4 Висновки до розділу 1 14

2. РОЗРОБКА ВИМОГ ДО СИСТЕМИ 16

2.1 Функціональні вимоги 16

2.2 Технічні вимоги 17

2.3 Висновки до розділу 2 18

3. ОПИС ВИКОРИСТАНОГО НАБОРУ ТЕХНОЛОГІЙ ТА ЗАГАЛЬНОПРИЙНЯТИХ ПРИНЦИПІВ СТВОРЕННЯ WEB-ЗАСТОСУНКІВ 19

3.1 Принципи проектування 19

3.1.1 Парадигма об’єктно-орієнтованого програмування 19

3.1.2 Принципи програмування S.O.L.I.D. 20
3.2 Огляд обраних мов програмування та фреймворка 23
3.2.1 Платформа .NET 23
3.2.2 Мова програмування C# 24
3.3 Клієнт-серверна архітектура 25
3.4 СУБД MSSQL, ORM система Entity Framework Core 26
3.5 Опис CQRS 28
3.6 WebAPI та засіб генерації документації Swagger 29
3.7 Спосіб авторизації 30
3.8 Веб-технології HTML, CSS 31
3.9 Мова програмування Java Script та фреймфорк React 32
3.10 Висновки до розділу 3 34
4 ОПИС ПРОЦЕСУ РОЗРОБКИ СИСТЕМИ 36
4.1 Реалізація клієнт-серверної архітектури 36
4.2 Реалізація принципу CQRS 39
4.3 Опис структури бази даних та сутностей 42
4.4 Створення авторизації в системі 47
4.5 Структура проекту клієнтської частини 49
4.6 Висновки до розділу 4 51
5 КЕРІВНИЦТВО ВИКОРИСТАННЯ СИСТЕМИ 52
5.1 Принцип роботи системи 52
5.2 Процес користування системою 54

5.3 Керівництво розробнику по інтеграції власного клієнтського застосунку з системою 57
5.3.1 Опис доступного API та документації до нього 57
5.3.2 Процес авторизації в системі 58
5.3.3 Моделі помилок 61
5.4 Висновки до розділу 5 62
ВИСНОВКИ 64
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 65
ДОДАТОК А .....
ДОДАТОК Б .....
ДОДАТОК В .....
ДОДАТОК Г .....
ДОДАТОК Д .....

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система;

IL – Intermediate Language;

CLR – Common Language Runtime;

ООП – Об’єктно-орієнтоване програмування;

ПЗ – програмне забезпечення;

СУБД – системи управління базами даних;

SQL – Structured Query Language;

T-SQL – Transact SQL;

DDL – Data Definition Language;

DML – Data Manipulation Language;

DCL – Data Control Language;

IDE – Integrated Development Environment;

ORM – Object-relational Mapping;

CQRS – Command Query Responsibility Segregation;

HTTP – Hyper Text Transfer Protocol;

JSON – JavaScript Object Notation;

JWT – JSON Web Token;

HTML – Hypertext Markup Language;

CSS – Cascading Style Sheets;

DOM – Document Object Model;

DI – Dependency Injection;

ВСТУП

Щороку суспільство прогресує та розвивається. Те що вчора здавалось новинкою, сьогодні вже буденна справа. В першу чергу цьому сприяє автоматизація процесів. Так заводи можуть випускати в рази більше продукції на відмінну від тих, які не автоматизували процес розробки. Це економить безліч часу та грошей.

Чинником, який безпосередньо впливає на швидкість розвитку держави, є якість навчання. Освіта завжди відігравала важливу роль в житті суспільства. Факт того, що освітній процес в нашій державі не автоматизований, здається досить дивним, оскільки в усіх інших сферах нашого життя для їхнього покращення вже давно використовуються машини. Крім цього, зважаючи на те, що в будь-якого школяра є сучасний смартфон або планшет, у кожної сім'ї є хоча б один ноутбук або комп'ютер, а доступ до Інтернету є в більшості місць, ідея використання системи моніторингу успішності в шкільних закладах виглядає досить актуальною.

Така система автоматизує задачі всіх учасників навчального процесу, що, безумовно, позитивно вплине на розвиток держави. Вчителя це позбавить від потреби контролювати наявність журналу, щоб виписати в нього оцінки, а носити їх взагалі не доведеться. Крім цього, доступ до щоденника будь-якого учня буде завжди.

Учнів така система також звільнить від паперових щоденників, які завжди доводиться носити з собою, та гарантує їм швидкий доступ до потрібної інформації в будь-який момент часу.

Батькам учнів дасть можливість контролювати успішність навчання та відвідування школи їхньою дитиною, витрачаючи на це менше часу та зусиль.

Ці причини змусили задуматись про створення системи моніторингу успішності. Слід зазначити, що такі системи вже успішно використовуються у вищих навчальних закладах, саме тому розглянута система призначена для шкільних закладів.

Система, яка буде описуватись в цій роботі матиме можливість для використання кінцевими користувачами, а також даватиме можливість розробникам створювати свої додати на її основі.

1. АНАЛІЗ СИСТЕМ МОНІТОРИНГУ УСПІШНОСТІ В ШКІЛЬНИХ ЗАКЛАДАХ

1.1 Актуальність системи для різних типів навчальних закладів

В шкільних закладах, як і у вищих навчальних відбувається процес навчання. Отже, логічним є припущення, що системи моніторингу успішності в них є спільні, проте це не так. Щоб вияснити чому, слід розглянути ці процеси в кожному з них.

В шкільних закладах звичайний робочий день включає від 5 до 8 уроків по 45 хвилин кожен. Всі вони одного типу, немає поділу на лекції або практики. В кінці кожного семестру, який триває половину навчального року, учні отримують оцінки, що підсумовують їхні показники протягом цього періоду. Інших аналогів процесу навчання я не знаю. Якщо такі є, то вони дуже індивідуальні.

Щодо вищих навчальних закладів, цей процес не настільки узагальнений. Кожен заклад визначає його індивідуально для себе, керуючись загальновідомими методиками. Таким чином один вищий навчальний заклад може мати систему, в якій потрібно проходити атестації посеред семестру, а вкінці здавати заліки та сесію. В той час як інший вищий навчальний заклад використовує наступну систему: цілий місяць вивчається лише одна дисципліна і по завершенні цього періоду здається залік. Також є кілька типів занять у вищих навчальних закладах: лекції та практики.

Через суттєву відмінність в організації навчання доцільним є створення системи моніторингу успішності окремо для кожного з типів навчальних закладів. Крім цього вищі навчальні заклади мають можливість власноруч реалізувати таку систему за необхідності. Таким чином у кожного з них є своя система моніторингу успішності навчання, яка підхо-

дить під його систему, а в шкільних закладів немає. Існують системи, які є спільні для всіх закладів шкільного типу, але вони в більшості платні. Через це більшість шкіл все ще не автоматизували процеси навчання і витрачають набагато більше часу. Саме тому в даній роботі створюється система моніторингу успішності для шкільних закладів.

1.2 Причини використання подібних систем

Будь-яке програмне забезпечення (ПЗ), створене не для розваг, основною метою має пришвидшення певного процесу. Система моніторингу успішності в шкільних закладах створена для полегшення навчального процесу та його оптимізації.

Оскільки способи використання системи відрізняються в залежності від типу користувача, тому й причини його використання різні. Розглянемо їх окремо для кожного типу.

Найвагомішою причиною використовувати системи моніторингу успішності в шкільних закладах для вчителів є можливість виставляти оцінки онлайн. Що у свою чергу означає, що їм більше не прийдеться очікувати журнал поки його заповнить інший вчитель. Адже будь-який вчитель зможе виконати необхідні операції з журналом в зручний йому момент без затримок. Крім цього, плюсом є те, що є доступ до журналів будь-якого класу.

Іншою перевагою системи моніторингу успішності в шкільних закладах є факт того, що вчителю більше не доведеться носити багато журналів, щоб їх заповнити. Всі вони будуть в електронному вигляді, що дасть змогу звільнити його від зайвого вантажу.

Найбільш очевидною перевагою є оптимізація роботи вчителя в результаті автоматизації певних його функцій. Таким чином, йому більше не доведеться формувати звіт про навчальний план на рік або ж витрача-

ти час на те, щоб повідомити батьків про відсутність їхньої дитини на уроці. Це також допоможе відповідному вчителю зекономити час на формуванні розкладу для школи. Процес який займав кілька години, тепер займатиме менше ніж хвилину. Це ж стосується і процесу складання рейтингу учнів школи.

Для учнів перевагою використання такої системи є факт того, що всі функції щоденника тепер будуть у смартфоні. Оскільки телефони всі зазвичай носять з собою, це означає, що щоденник завжди буде доступний. Окрім щоденника, учень також матиме можливість будь-коли переглянути свої оцінки, які виставив учитель. Для цього йому не потрібно буде очікувати уроку, як це було раніше.

Учням більше не доведеться піклуватись про те, щоб записувати домашнє завдання. Оскільки воно буде доступне в системі після його публікації вчителем. Після виконання завдання буде можливість завантажити його в систему, де вчитель зможе перевірити завдання онлайн. Це позбавить необхідності приносити домашнє завдання до школи, а також дасть можливість вчителю зробити перевірку в будь-який зручний йому час.

Для батьків учнів перевага у використанні системи моніторингу успішності також очевидна. Вони зможуть дізнаватись оцінки дитини будь-коли онлайн, не витрачаючи час на те щоб приїхати до школи й знайти потрібного вчителя. А у випадку відсутності їхньої дитини на уроці, вони будуть обов’язково сповіщені. Окрім цього батьки матимуть доступ до щоденника їхньої дитини, тобто зможуть бачити всі зауваження та своєчасно на них реагувати. В загальному система моніторингу успішності в шкільних закладах дає можливість пришвидшити роботу в школі, ти самим покращити рівень освіти в країні.

1.3 Огляд існуючих рішень для моніторингу успішності в шкільних закладах

Щоб повністю проаналізувати систему моніторингу успішності в шкільних закладах і врахувати переваги існуючих рішень й уникнути недоліків, на основі [11] було порівняно деякі найбільш популярні з них.

1.3.1 Шкільна освітня мережа «Щоденник.ua»

Шкільна освітня мережа «Щоденник.ua», зображена на рисунку 1.1, безкоштовна через її підтримку Міністерством освіти та науки, що, безумовно, є її перевагою над аналогами. Деякі сторінки мережі занадто перенавантажені інформацією, що робить роботу користувача з нею трохи складнішою. Також слід зазначити, що дизайн системи не є сучасним, що також негативно впливає на роботу з нею.

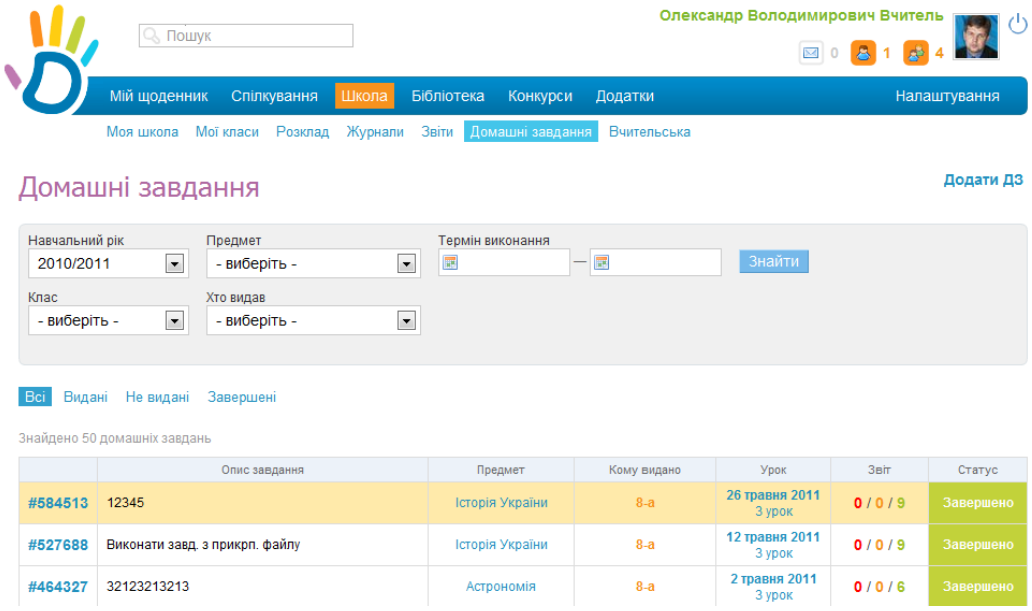


Рисунок 1.1 – Шкільна освітня мережа «Щоденник.ua»

Щодо функціональності, то весь основний функціонал в ній присутній. Є розклад уроків, електронний журнал та щоденник. Крім цього в ній наявний додатковий функціонал. Частина з нього є актуальною: повідомлення, можливість завантаження файлів, але також присутній функціонал, який є надлишковим: власний перекладач, можливість розміщувати шкільну газету, можливість створювати групи. В загальному, принцип роботи системи зрозумілий для кінцевих користувачів.

Оскільки мережа представлена у вигляді веб застосунку, доступ до неї встановлюється за допомогою браузера. Перевагою даного освітньої мережі є можливість безкоштовного використання та зрозумілий принцип роботи, а недоліками – застарілий інтерфейс та його перенавантаження даними, що ускладнює користування нею. Також негативно впливає на досвід використання мережі надлишковий функціонал.

1.3.2 Система «Електронний журнал»

Система «Електронний журнал», зображена на рисунку 1.2, має сучасний дизайн та не містить зайвої інформації.

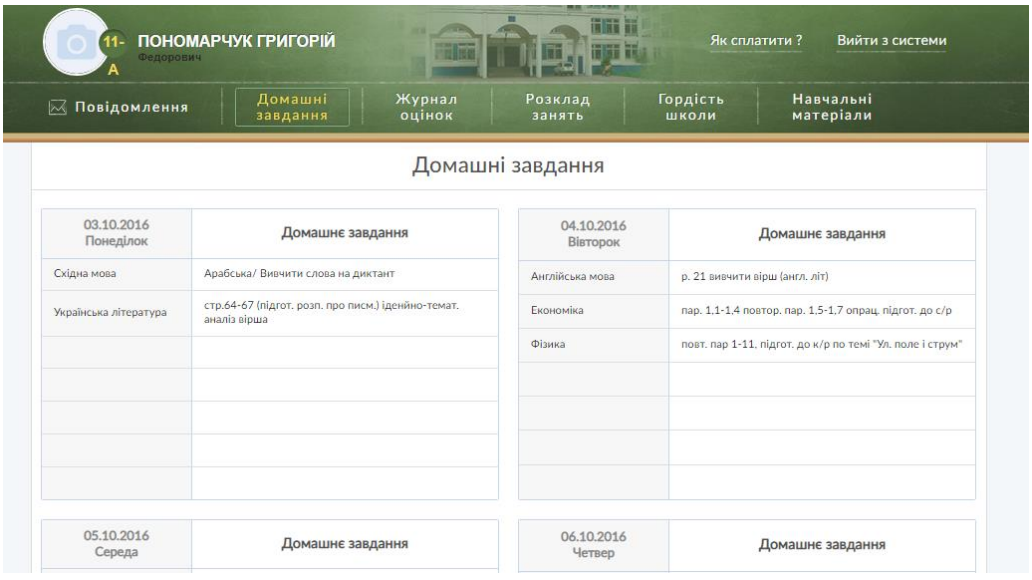


Рисунок 1.2 – Система «Електронний журнал»

Це значно полегшує користування. Однак вагомим недоліком є факт того, що ця система є платною. Місяць використання системи буде коштувати навчальному закладу 499 грн. За семестр слід заплатити 1345 грн, а за рік – 1999 грн. Слід зазначити, що ціни досить прийнятні. Весь основний функціонал присутній.

Окрім нього, ця система містить актуальні додаткові можливості: повідомлення, бібліотека, рейтинг учнів в школі, автоматичне формування звітів для вчителя. Проте не всі додаткові послуги автоматизовані. Наприклад, послуга, SMS-сповіщення батьків у разі прогулювання занять їхньою дитиною потребує додаткової роботи зі сторони оператора школи.

1.3.3 Освітній портал «Класна оцінка»

Ще одним аналогом для моніторингу успішності в школах є портал «Класна оцінка», веб-інтерфейсу якого зображено на рисунку 1.3.

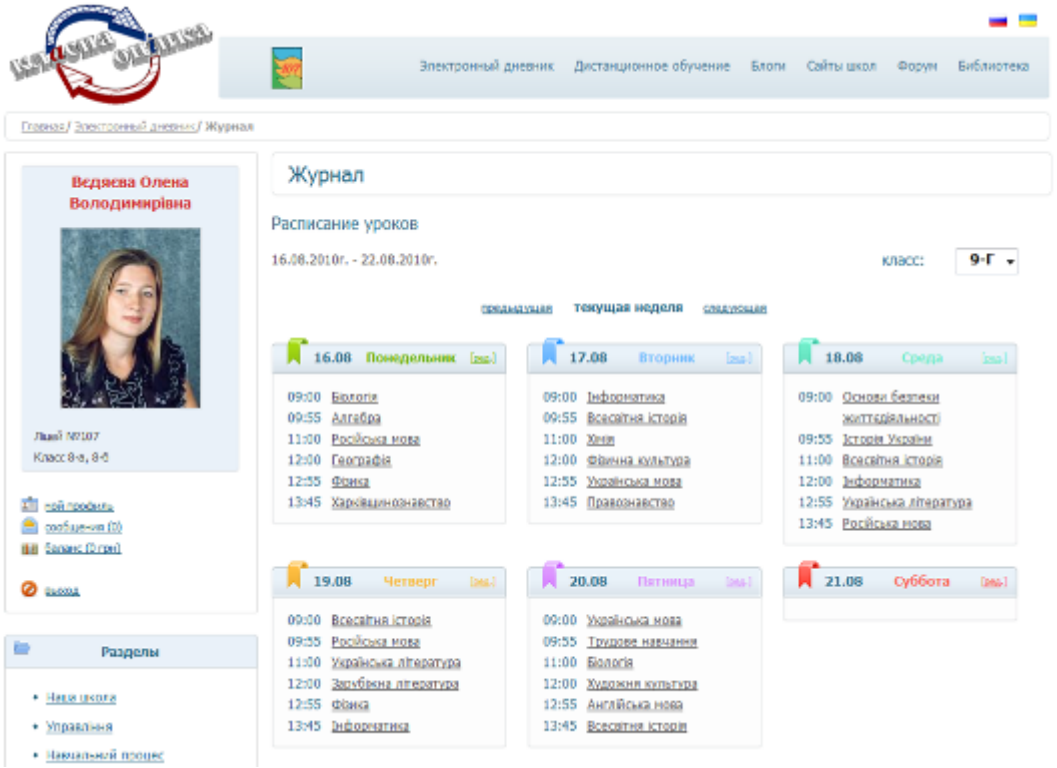


Рисунок 1.3 – Освітній портал «Класна оцінка»

Його дизайн досить приємний, але не дуже сучасний. Зважаючи на те, що сторінки трохи перенавантажені надлишковою інформацією користуватись ним не зручно. Це є одним з недоліків системи «Класна оцінка». Нажаль, користування цією системою платне. Існує 3 тарифи: «сайт», «сайт+», «професіонал». Вартість тарифу «сайт» становить 550 грн на рік. За користування тарифом «сайт+» треба заплатити 700 грн на рік, а за «професіонал» – 900 грн на рік. Оскільки ця система являє собою освітній портал, то в ній є безліч функцій, але придатні до використання у шкільних закладах є лише в одному тарифі. Наприклад, функція «щоденник» доступна лише в тарифі «професіонал». І оскільки ця функція входить в перелік основного функціоналу, то слід орієнтуватись на ціну саме цього тарифу.

Щодо функціональності, в цій цьому освітньому порталі є досить багато можливостей. Окрім основної функціональності є багато додаткової. В цій системі є можливість обмінюватись повідомленнями, створювати блоги, форуми, різноманітні опитування. Також є багато функцій для організації онлайн навчання. Наприклад, електронна бібліотека з необхідними підручниками, наявність обміну повідомлень між учасниками навчального процесу та можливість створювати відеоконференції. Надлишковою є функціональність створення власного сайту школи. Для цієї додаткової можливості виокремили занадто багато конфігурації. Це може заплутати представника школи, який замість того, щоб приділяти більше уваги налагодженню учбового процесу, буде займатись створенням сайту.

1.4 Висновки до розділу 1

В підрозділі 1.1 доведено актуальність системи моніторингу успішності для різних типів навчальних закладів. В результаті аргументовано

необхідність системи моніторингу успішності саме для шкільних навчальних закладів.

В підрозділі 1.2 проведено аналіз причин використання таких систем для кожного типу користувачів. Для вчителів – економія часу, для учнів – доступ до потрібної інформації онлайн, а для батьків – легкість контролю над дитиною.

В підрозділі 1.3 порівняно 3 існуючих рішення моніторингу успішності в шкільних закладах. В результаті визначено їхні основні переваги та недоліки. Серед переваг слід виокремити зручний інтерфейс використання, а основним недоліком є зайва функціональність, яка ускладнює користування системою.

2. РОЗРОБКА ВИМОГ ДО СИСТЕМИ

2.1 Функціональні вимоги

З огляду існуючих рішень можна виокремити основну функціональність, яка присутня в кожній з систем та дає можливість перевести в електронний вигляд всі необхідні дані, таким чином оптимізувавши роботу у школі. До переліку таких функцій входять:

- Розклад уроків – у користувачів системи повинна бути можливість переглянути розклад уроків будь-якого класу або вчителя. Інформація повинна бути зображена в зручному для користувача вигляді та містити дані про час, місце проведення уроку, вчителя та клас;
- Електронний журнал – основна функція системи. Учням вона дає можливість переглядати свої оцінки з будь-якого предмету. Перегляд оцінок інших учнів повинен бути не доступним. Вчитель повинен мати можливість переглядати оцінки всіх учнів та виставляти оцінки тільки в тих класів і тільки з тих предметів, які він викладає;
- Електронний щоденник – функціональність системи, яка дає забезпечує всі можливості щоденника онлайн. Вчителі повинні мати змогу виставляти оцінки учням та писати зауваження. Учні мають можливість переглядати їх. Повинен бути фільтр, щоб переглядати записи за певний період;
- Домашні завдання – це функціональність, що дає змогу вчителям вказувати інформацію про домашні завдання та перевіряти їх. Учні повинні мати змогу переглядати домашні завдання завантажувати файли з виконаним домашнім завданням.

Крім функціональності, яка дасть змогу контролювати необхідні дані, також слід реалізувати наступні функції, які автоматизують певні процеси:

- Генерація звітів для вчителя – повинна бути можливість автоматично згенерувати звіт з даними про навчальний план. Вхідними даними для генерації звіту є перелік тем уроків, внесених в систему під час навчального року;
- SMS сповіщення – функціональність посилання SMS повідомлення батькам учня у разі пропуску ним уроку ;
- Генерація розкладу – повинна бути можливість автоматично згенерувати розклад уроків для всіх класів. Вхідними даними для цього є інформація про кількість уроків на тиждень кожного предмету для кожного класу, присутнього в школі;
- Рейтинг учнів – система повинна мати можливість генерувати звіт про успішність учнів, в якому показувати учнів та їх середні бали зі всіх предметів. Учні повинні бути посортовані в порядку спадання середнього балу. Також повинна бути можливість фільтрувати результати.

2.2 Технічні вимоги

Однією з особливостей системи є можливість створити власний клієнтський застосунок. Тому, щоб це реалізувати, слід зробити API, який дасть змогу виконувати всі необхідні функції системи. Крім його реалізації повинна бути документація, яка б описувала всі методи взаємодії. Така документація може бути створена за допомогою використання засобів автоматичної генерації документації.

Оскільки API буде використовуватись іншими розробниками для створення власних застосунків, повинні бути моделі помилок, які легко обробляти та містять інформативні повідомлення причини помилки.

Обов’язковою умовою функціонування є безпечність системи. Повинен використовуватись сучасний та надійний спосіб авторизації користувачів в системі.

Архітектура системи повинна бути такою, щоб в майбутньому можна було з легкістю розширювати її. Так, наприклад, планується створити можливість зазначення класних кімнат представником школи при її формуванні.

Реалізована клієнтська частина системи повинна мати мінімалістичний дизайн, який не міститиме зайвої інформації та не буде перешкоджати користувачу взаємодіяти з системою. Також вона повинна буде показувати високу швидкодію. Для цього дані, які рідко змінюються повинні зберігатись в кеші, для прискорення роботи системи.

2.3 Висновки до розділу 2

В розділі 2 проведено огляд основних вимог до системи. Їх розділено на два типи: функціональні та технічні. Основні функціональні вимоги описано в підрозділі 2.1. Найбільш значущою з них є перегляд розкладу онлайн. Технічні вимоги зазначено в підрозділі 2.2. Серед них можна виокремити можливість створення власних клієнтських застосунків за допомогою документованого WebAPI.

3. ОПИС ВИКОРИСТАНОГО НАБОРУ ТЕХНОЛОГІЙ ТА ЗАГАЛЬНОПРИЙНЯТИХ ПРИНЦИПІВ СТВОРЕННЯ WEB-ЗАСТОСУНКІВ

3.1 Принципи проектування

Існує безліч систем та застосунків. Вони відрізняються складністю, призначенням та кількістю функціональності, яку містять в собі. Та незалежно від цього є перелік принципів, яких слід дотримуватись при проектуванні будь-яких систем. Вони допоможуть підтримувати зрозумілу структуру проекта, зроблять процес внесення змін легким й швидким та зменшать ймовірність виникнення помилок після нього.

3.1.1 Парадигма об’єктно-орієнтованого програмування

Об’єктно-орієнтована парадигма програмування – це один з існуючих способів мислення розробника та стиль написання програм в основі якої лежать об'єкти. Вони визначають в собі певну обмежену логіку, що дає змогу чітко визначити інтерфейс взаємодії з ними. ООП має три основних принципи: інкапсуляцію наслідування та поліморфізм.

Інкапсуляція забезпечує безпечну взаємодію з об’єктом класу. Це реалізовується за допомогою визначення відкритого інтерфейсу взаємодії, який складається з атрибутів та методів класу, які можна використовувати за його межами. Правильно визначений відкритий інтерфейс класу забезпечує меншу ймовірність його неправильного використання.

Принцип наслідування дає можливість створювати ієрархії класів, що допомагає уникнути дублювання коду таким чином допомагає зекономити безліч час та зменшити кількість коду в класі. Наслідування дає

змогу описати новий клас за допомогою вже існуючих, якщо він включає в себе ті ж методи, які присутні в класі від якого унаслідується.

Ще один принцип без якого важко писати в стилі ООП – поліморфізм. Досить точно його зміст описує фраза «Один інтерфейс, безліч реалізацій». Тобто поліморфізм дає можливість зробити один інтерфейс, в якого може бути скільки завгодно реалізацій, що в свою чергу робить можливим послабити зв’язки між класами. Інтерфейс являє собою певну абстракцію, яка містить контракт взаємодії з ним для навколишнього світу. Буд-який клас, що реалізовує потрібний інтерфейс, може реалізовувати його по-своєму. А оскільки контракт взаємодії буде збережено, то це дасть змогу легко змінювати поведінку системи, додаванням нового коду та не змінюючи вже існуючого. Загалом використання поліморфізму робить систему дуже гнучкою до змін, що дає наступні переваги: зменшення ймовірності зробити помилку в системі та можливість швидко додавати нову функціональність.

3.1.2 Принципи програмування S.O.L.I.D.

Окрім основних принципів ООП існують загальновідомі принципи, які позитивно впливають на створення програмного забезпечення. Найвідомішими з них є принципи S.O.L.I.D, де назва – абревіатура перших букв кожного з принципів, який до них входить. Вона розшифровується наступним чином:

- S – Single responsibility principle;
- O – Open Closed Principle;
- L – Liskov Substitution Principle;
- I – Interface Segregation Principle;
- D – Dependency Inversion Principle.

Single responsibility principle (SRP) – принцип єдиної відповідальності говорить про те, що клас повинен створюватись лише з однією метою. Іншими словами, він повинен мати лише одну причину для змін. Першою перевагою дотримання цього принципу є те, що він робить код більш зрозумілим. Також він робить процес внесення змін набагато швидшим, оскільки логіку, яку потрібно буде змінити, зберігається лише в одному місці. Крім цього, він зменшує ймовірність внесення помилки.

Open/closed principle (OCR) – принцип відкритості закритості є також дуже важливим принципом, суть якого описується наступним твердженням. Клас повинен бути відкритий для розширення, проте недоступний для модифікації. Це означає, що за необхідності внесення змін в певний клас, розробник повинен додавати нову функціональність, яка буде розширювати вже існуючу, а не змінювати поведінку старої. Як і в попередньому принципі, дотримання цього принципу робить систему гнучкою до змін. Оскільки існуюча функціональність не змінюється, тобто вона буде працювати як і очікувалось раніше, то зменшується ймовірність внесення помилки. Також це робить проект більш зрозумілим та прискорює внесення змін в нього. Адже зміни потрібно буде внести лише в одному місці, а не в кількох, як це буває при зміні вже існуючої функціональності. Цей принцип реалізовується за допомогою наслідування та поліморфізму, шляхом перевизначення старої функціональності в класах наслідниках.

Liskov Substitution Principle (LSP) – принцип підстановки Лісков. Зрозуміти його досить важко, проте необхідно. Суть цього принципу наступна: функції, які використовують батьківський тип, повинні мати змогу замінити його на дочірній, не знаючи про це. Тобто, якщо є місце в проекті, де використовується поліморфізм і замість базового класу, який прописаний в сигнатурі метода, передається дочірній, помилки виникнути не повинно. Це дає можливість використовувати поліморфізм безпеч-

ною. Дотримування цього принципу робить поведінку програми більш очікуваною та логічною, що зменшує час, необхідний новому розробнику, щоб розібратись з проектом. Також він зменшує ймовірність внесення помилки.

Interface Segregation Principle – принцип поділу інтерфейсів. Його суть описується наступним твердженням: жоден клієнт не повинен реалізовувати методи, які він не використовує. В даному випадку клієнтом є клас, який реалізує інтерфейс. Цей принцип дуже зв'язаний з SRP, адже якщо інтерфейс буде створений лише з однією метою, то він міститиме перелік методів, які його клієнти будуть використовувати. Якщо є випадок, коли клієнт А використовує всі методи, зазначені в інтерфейсі, а клієнт Б ні, то доцільніше розбити інтерфейс розділити на кілька і в клієнті А реалізувати всі з них, а в клієнті Б лише, той, який містить метод, що використовується. Дотримання цього принципу можливе через те, що всі сучасні мови програмування забезпечують реалізацію класом кількох інтерфейсів. Він дає змогу легше дотримуватись SRP, тим самим зменшуючи можливість внесення помилки, процес змін набагато швидшим, а проект більш зрозумілим.

Dependency Inversion Principle – принцип інверсії залежностей. Залежностями є використання одних класів в інших. Його зміст складається з двох частин. Перша – класи високого рівня не повинні залежати від класів низького рівня, обидва повинні залежати від абстракцій. Друга – Абстракції не повинні залежати від деталей, деталі повинні залежати від абстракцій. Абстракціями в даному випадку вважаються інтерфейси, які описують контракти взаємодії тих, хто їх реалізує. Тобто перша частина говорить про те, що класи високого та низького рівня в якості залежності повинні мати інтерфейси. Друга частина говорить про те, що ці інтерфейси не повинні залежати від конкретних класів, а також від інтерфейсів. Дотримання цього принципу дає змогу легко вносити зміни в систему.

Для редагування поведінки потрібно створити реалізацію інтерфейсу, який очікується в якості параметра, та використати його в потрібному місці. Ця перевага дуже корисна при написанні юніт-тестів. Також цей принцип пришвидшує розробку шляхом інкапсуляції логіки, яка відповідає за реєстрацію конкретного класа певному інтерфейсу.

3.2 Огляд обраних мов програмування та фреймворка

Для реалізації серверної частини системи було вирішено використовувати мову програмування C#. Ця мова входить до платформи .NET. Веб-застосунок типу WebAPI було реалізовано на ASP.NET Core. Також, для уникнення дублювання логіки, загальну логіку було винесено в Nuget пакети, які використовувались у потрібних рішеннях.

3.2.1 Платформа .NET

Платформа .NET, створена відомою компанією Microsoft, на сьогоднішній день складається з двох фреймворків: .NET Framework та .NET Core. Перший був створений на багато раніше для написання програм під ОС Windows. Пізніше великої популярності набула розробка веб-застосунків для цього компанія Microsoft випустила фреймворк ASP.NET, який все ще був заточений під ОС Windows. Оскільки вартість серверів на цій ОС є на порядок вище ніж на інших стала актуальною потреба писати застосунки, які б не залежали від ОС. Рішенням став новий фреймворк від компанії Microsoft – .NET Core. Він з’явився лише 2 роки тому, проте дуже стрімко розвивається. При його розробці намагались дотримуватись тої ж архітектури, що і в .Net Framework. Принцип його роботи залишився схожим: код, написаний мовою програмування високого рівня, компілюється в проміжковий на IL (Intermediate Language). Після цьо-

го цей код виконується в CoreCLR (Common Language Runtime). (В .NET Framework IL коди виконується в CLR). Безумовно головною перевагою .NET Core над .NET Framework є можливість виконання на різних платформах, проте це не єдина. Також слід відзначити, що коди .NET Core знаходиться у відкритому доступі, що часто допомагає зрозуміти як це працює всередині.

3.2.2 Мова програмування C#

Найбільш популярною мовою програмування на платформі .NET є C#. Вона є зрозумілою та легкою для вивчення. Ця мова програмування є об’єктно-орієнтованою, що означає, що в ній наявні всі принцип ООП. В ній присутня строга типізація, тобто тип змінної не може бути змінений в ході виконання програми. Також C# має статичну типізацію, що означає, що перевірка відповідності типів даних та сигнатур відбувається на моменті компіляції, а не в ході виконання програми.

3.2.3 Технологія ASP.NET Core

Технологія ASP.NET Core призначена для створення веб-застосунків. Вона була представлена одночасно з виходом нової платформи .NET Core і вже сьогодні доступна її бета-версія 3.0. Це свідчить про її швидкий розвиток та попит на неї у світі розробників програмного забезпечення. Ключовою відмінністю від попередника – ASP.NET, є можливість працювати на різних ОС. Код з вихідним кодом доступний у відкритому доступі. Необхідні компоненти для створення веб застосунків доступні в якості окремих модулів в Nuget. З допомогою цього фреймворку можна створювати веб застосунки різних типів: MVC, WebAPI. Його

було обрано через високу продуктивність, незалежність від ОС та наявність великої кількості матеріалу в інтернеті по ньому.

3.2.4 Менеджер пакетів Nuget

Nuget – менеджер пакетів для платформи .NET. Nuget- являє собою звичайний архів, з розширенням .nupkg. В ньому зберігається скомпільований код проекту. Метою його використання є потреба виокремити код, який часто використовується, в окрему бібліотеку, щоб уникнути його дублювання.

3.3 Клієнт-серверна архітектура

Клієнт-серверна архітектура – архітектура програмного забезпечення, яка складається з централізованої серверної частини та клієнтської, яка виконується на машинах клієнтів. Принцип роботи такої архітектури наступний: на серверній частині формується інтерфейс взаємодії з нею, клієнтська частина використовує його для отримання або посилання відповідної інформації. В серверній частині знаходиться більша частина логіки застосунку, наприклад, робота з базою даних, бізнес логіка. Переважно інтеграція з сторонніми системами відбувається також тут. Клієнтській частина містить логіку для відображення даних та слугує інтерфейсом взаємодії користувача з системою.

Однією з переваг такої архітектури є можливість створювати кілька клієнтських частин для однієї серверної. При чому клієнтські частини можуть працювати на різних ОС та платформах. Це дає змогу пришвидшити розробку ПЗ, оскільки не потрібно створювати серверну частину щоразу. Також системи з такою архітектурою мають досить високу продуктивність. Це зумовлено тим, що на клієнтській частині, яка викону-

ється на машинах користувачів, не виконується складних, об’ємних дій. Вони відбуваються на серверній частині системи, для якої зазвичай використовують машини потужніші ніж у користувачів.

3.4 СУБД MSSQL, ORM система Entity Framework Core

В якості системи управління базами даних (СУБД) було використано MSSQL Server. Ця СУБД, створена компанією Microsoft, використовує різновидність мови Structured Query Language (SQL), яка називається Transact-SQL. В цій мові команди поділяються на 3 типи:

- команди, які визначають дані при їх створенні відносять до групи Data Definition Language (DDL);
- команди, за допомогою яких ми можемо керувати даними, наприклад, додавати, редагувати або видаляти, відносяться до Data Manipulation Language (DML);
- команди, що керують доступом до даних, відносяться до типу Data Control Language (DCL).

Основними перевагами даної СУБД є продуктивність та швидкість роботи, надійність та безпечність, яка забезпечується шляхом шифрування даних, та простота роботи з нею. Слід додати, що існує розширення для Integrated Development Environment (IDE) Visual Studio, в якому й велась розробка, який дає можливість в середині IDE керувати базами даних за допомогою графічного інтерфейсу. Це допомагає зрозуміти де і які дані зберігаються та пришвидшує розробку.

Однією з особливостей MSSQL Server порівняно з іншими СУБД є можливість відмінити запит посеред його виконання. Можливість виконати запити на кілька баз даних одночасно також є особливістю цієї СУБД. Ще однією відмінністю є можливість створювати колонки, значення яких автоматично розраховується за певною формулою.

Для взаємодії з базами даних в .NET використовують бібліотеку ADO.NET. Хоча вона дає можливість гнучкого налаштування та опису взаємодії, це займає досить багато часу. Тому використовують технологію Object-relational Mapping (ORM). В .NET найбільш популярною ORM технологією є Entity Framework. Ця технологія доступу дає можливість взаємодіяти з даними програмним шляхом. Необхідність писати SQL запити різко знижується, адже Entity Framework генерує їх на основі коду. Це значно пришвидшує та полегшує розробку системи. В Entity Framework є два підходи інтеграції з системою. Перший називається code first, та використовується при створенні нової бази даних та генерації таблиць на основі сутностей, визначених в коді. Інший підхід називається database first та працює в протилежному напрямі. На основі вже існуючої бази даних генерується клас моделей, які слід використовувати в коді для взаємодії з таблицями.

Принцип роботи цієї технології полягає в наступному. Створюються класи, які описують таблиці, їх називають сутностями. Після цього створюється клас, який буде контекстом бази даних та міститиме дані про таблиці та даватиме можливість конфігурувати взаємодію з базою даних. В конфігурації створюється рядок підключення до бази даних. Після цього за допомогою відповідної команди автоматично генерується код, який описує зміни в базі даних, це називається міграцією. В Entity Framework такий код розміщується в автоматично згенерованому класі. Останнім кроком є застосування цих змін за допомогою відповідної команди.

Для .NET Core був створений, як і попередній компанією Microsoft, Entity Framework Core. Компанія зуміла зберегти інтерфейс взаємодії з ним майже незмінним, за виключенням кількох команд.

3.5 Опис CQRS

В якості архітектурного підходу було використано Command Query Responsibility Segregation (CQRS). З назви можна зрозуміти, що його суть полягає у виокремленні команд та запитів. В даному контексті командою вважають те, що змінює стан системи, а запитом – те, що лише повертає дані про систему, не змінюючи її. Переважно такий підхід застосовують для взаємодії з базою даних. Принцип роботи цього підходу сформовано у [5] та зображено на рисунку 3.1.

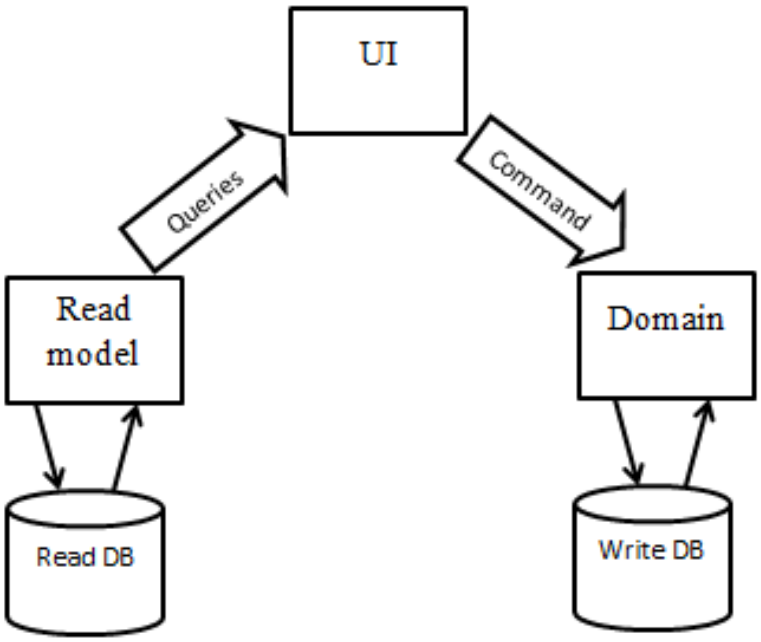


Рисунок 3.1 – Принцип роботи CQRS

На основі вхідних даних формуємо команду або запит, які містять лише необхідну інформацію для їх виконання. Після цього за допомогою відповідного обробника виконуємо запит або команду та у випадку запиту отримаємо результат. Такий підхід легкий в реалізації, проте на почат-

ку проекта потрібно відносно багато часу для його реалізації. Тому його доцільно використовувати на проектах середнього та великого розмірів.

Використання CQRS дає змогу швидко вносити зміну в систему або редагувати вже існуючу логіку. Причиною цього є те, що вся необхідна логіка інкапсульована в одному місці і її не потрібно шукати по цілому проекту. Також такий підхід легкий для розуміння, що скорочує час, необхідний новому розробнику для розуміння роботи системи.

3.6 WebAPI та засіб генерації документації Swagger

В сучасному світі розробки ПЗ популярним є використання підходу WebAPI для інтерграції з сторонніми системами або для використання власних. Тобто, WebAPI надає контракти взаємодії з системою. В цілому WebAPI складається з багатьох точок взаємодії з них, які називаються кінцевими точками. Доступ до кожної з кінцевих точок відбувається через Hyper Text Transfer Protocol (HTTP). В цьому протоколі кожна точка має власну адресу, за якою можна зробити запити до неї, один з визначеного переліку методів, який описує тип дії, яку виконує запит та параметри, які є необов'язковими. Відповідь на HTTP запит містить тіло відповіді, яке може бути пустим або повертати необхідну інформацію та статус запиту, який свідчить про успішність його виконання.

Спосіб взаємодії через WebAPI є неможливим без документації, яка описує вищезгадані кінцеві точки. Така документація необхідна як для сторонніх користувачів, які прагнуть інтегруватись з системою, так і для співробітників компанії, які використовують відповідну частину системи. Раніше така документація створювалась співробітниками компанії. Однак тепер існує можливість автоматизувати цей процес. Це досягається шляхом автоматичної візуалізації контрактів взаємодії з системою через дос-

тупні кінцеві точки. Найбільш популярним рішенням на даний момент є Swagger.

Swagger – інструмент, який візуалізує контракти взаємодії з системою з внесенням мінімальних змін в код. Він доступний для використання з різними технологіями. Результат його роботи може бути представлений за допомогою веб-сторінки або в JavaScript Object Notation (JSON) форматі.

Очевидною перевагою використання Swagger є пришвидшення розробки системи шляхом зменшення часу, необхідного для написання документації. Іншим шляхом економії часу за допомогою цього інструменту, є використання його при тестуванні системи. Також він вказує на помилки при створенні контрактів взаємодії, що також пришвидшує розробку.

3.7 Спосіб авторизації

Одним з найбільш популярних сьогодні способів авторизації користувачів в системі є стандарт OAuth 2.0. Алгоритм його роботи досить зрозумілий та легкий для реалізації. В ньому користувач вводить необхідну інформації для його ідентифікації, яка передається на сервер. Там ці дані використовуються для формування access token'у, за допомогою якого можна ідентифікувати користувача в майбутньому. Access token повертається у веб-застосунок, з якого прийшов попередній запит. Всі наступні запити, що вимагаються авторизації користувача, будуть виконуватись з отриманим access token’ом. Зазвичай token розміщують в заголовку запита.

В якості access token’а часто використовують JSON Web Token (JWT). За [6] JWT – JSON об’єкт, який визначений у відкритому стандарті RFC 7519. Кожен JWT складається з заголовку, корисних даних та під-

пису. Заголовок та корисна частина записується у вигляді JSON об'єктів. Алгоритм створення JWT наступний: кожен з байтів, які відповідають символам в заголовку та корисній частині представляють в системі числення з основою 64. Після цього результат дії над заголовком та корисною частиною об’єднують, попередньо додаючи символ крапки між ними. Далі знову додають символ крапки та секретний ключ, який повинен бути доступний тільки в місті формування token’а. В кінці дані змінюють за допомогою одного з доступних алгоритмів і в результаті отримують рядок.

Для ідентифікації користувача перевіряють отриманий JWT та якщо token проходить валідацію, можна отримати дані, які були закодовані. Вони також будуть доступні як JSON об’єкти заголовка та корисної частини, в яких буде інформація, яка була туди записана на стадії створення цього JWT.

3.8 Веб-технології HTML, CSS

Для створення веб-сторінок, які бачить користувач використовують Hypertext Markup Language (HTML). HTML являє собою мову розмітки, яка використовується для відображення і структурування веб-сторінки. Вона складається із серії елементів, в яких розміщуються різні частини контенту, щоб відобразити їх відповідним чином. Елемент складається з початкового тегу, контенту та закриваючого тегу. Також елементи можуть мати атрибути, які описують їх. Вони розташовуються у відкриваючому тезі. Деякі елементи не містять контенту, тому їх називають порожніми елементами. Вони складаються лише з відкриваючого тегу та атрибутів. Кожна веб-сторінка повинна містити наступні теги: <!DOCTYPE html> – описує тип документа, <html> – використовується як кореневий елемент, <head> – інформація про сторінку, яка не буде відображатись як

її зміст, а використовуватиметься пошуковими системами, <body> – елемент, який містить весь контент, що відображається на сторінці.

HTML це не єдина технологія, яка використовується для створення сучасних веб сторінок. З її допомогою лише створюється та структурується контент сторінки, а те як має виглядати сторінка задається за допомогою Cascading Style Sheets (CSS).

CSS – мова стилів, яка має стандартизовану специфікацію W3C. Вона описує як елементи повинні відображатись. На даний момент існує 3 версії цієї мови: CSS1 вже застаріла, CSS2.1 рекомендована для використання та CSS3 – найновіша версія, яка є розбитою на модулі. З її допомогою можна задавати стилі для вибіркових елементів. Щоб виокремити елементи використовують атрибут class. Також можна ідентифікувати елемент за допомогою атрибуту id, проте більш правильно користуватись атрибутом клас. Щоб вказати до якого саме елементу застосовуються стилі його описують за допомогою селекторів. Для посилання на тег в селекторі слід зазначити його назву, для класу слід вказати назву перед якою додати символ «.», а для ідентифікаторів слід вказати символ «#» після якого написати сам ідентифікатор.

З метою уникнення дублювання коду та пришвидшення процесу написання стилів використовують препроцесори. Вони компілюють код написаний з використанням їхнього синтаксису в звичайний css код. Основне пришвидшення при написанні стилів з їх використанням забезпечується можливістю вкладати стилі один в одного.

3.9 Мова програмування Java Script та фреймфорк React

Для забезпечення інтерактивності сторінки та надання користувачу можливості взаємодіяти з нею використовують скрипти, написані мовою програмування JavaScript. ЇЇ можна застосовувати для різних програм:

серверні частини, мобільні застосунки та найчастіше вона використовується для написання клієнтської частини веб-застосунків. Стандартом для JavaScript є ECMAScript. На даний момент найбільш поширеною є версія ECMAScript 5.1, проте досить швидко починають дотримуватись стандарту ECMAScript 6. В цій мові програмування присутня динамічна типізація, що означає, що перевірка відповідності типів відбувається в процесі виконання скрипта. JavaScript має слабку типізацію, що означає, що змінна може змінювати свій тип в залежності від значення, яке їй присвоєно.

Взаємодія з інтерфейсом користувача відбувається через Document Object Model (DOM). В цій моделі сторінка є деревом, яке складається з вузлів, які представлені тегами. За допомогою відповідних методів з дерева можна дістати потрібний вузол та змінити його. Проте, щоб це зробити необхідно написати немало коду. Щоб вирішити цю проблему існує безліч JavaScript бібліотек, які покликані полегшити та пришвидшити процес роботи з DOM. Одним з найбільш популярних сьогодні є бібліотека створена компанією Facebook – React.

В цій бібліотеці веб-сторінка складається з багатьох компонентів. Вони можу бути вкладеними один в одного або використовувати всередині інші компоненти. Основне завдання кожного з них є ізолювати логіку певної частини сторінки від іншої, не зв’язаної з нею. Компоненти зображуються як теги. Кожен компонент являє собою клас, в якому обов’язково є метод render. Цей метод повертає HTML розмітку, яку буде відображено на веб-сторінці там, де викликається цей компонент. Також компоненти мають властивості та стан, які доступні за допомогою властивостей props та state відповідно. Props використовується для передачі параметрів від батьківського компонента дочірньому. Під час виклику компонента параметри, які треба передати в нього слід вказати як атрибути. Стан компонента ініціалізується в середині нього. Для його зміни слід використовувати метод setState, в який в якості параметра передати

об’єкт, що описуватиме стан компонента. Виклик цього метода виконає перевірку того, чи змінився стан компонента, і якщо так, то перемалює компонент шляхом виклику методу render з оновленими властивостями props та state.

Однією з переваг цієї JavaScript бібліотеки є те, що в разі зміни стану компонента не буде змінюватись весь DOM, а лише та частина, де відбулися зміни. Це робить React ефективним. Іншою перевагою цієї бібліотеки є простота її використання. Слід зазначити, що існує багато документації по ній.

3.10 Висновки до розділу 3

В підрозділі 3.1 було проведено огляд основних принципів проектування систем ПЗ, а саме ООП та SOLID. Визначено суть кожного з них та переваги, які отримує розробник у випадку їх дотримання.

В підрозділі 3.2 проведено огляд .NET Framework та .NET Core, описано принцип їхньої роботи та зазначено переваги їхнього використання. Також було виявлено особливості мови програмування C#. Було проаналізовано технологію для створення веб-застосунків ASP.NET Core. В результаті було зазначено причини її вибору для реалізації серверної частини застосунку. Було описано менеджер пакетів для платформи .NET – Nuget.

Архітектура, яка використовувалась при розробці системи описана в підрозділі 3.3. В ньому зазначено принцип функціонування за клієнт-серверної архітектури. Визначено переваги які вона дає розробникам та бізнесу в цілому.

В підрозділі 3.4 проведено огляд технологій, які використовувались для роботи з даними в системі. В результаті було описано СУБД MSSQL, розглянуто її особливості та проведено порівняння з іншими існуючими

СУБД , в ході якого визначено переваги MSSQL над аналогами. Також було розглянуто ORM систему Entity Framework. Описано принцип її роботи та вказано переваги, які вона надає.

Підхід CQRS було проаналізовано в підрозділі 3.5. В результаті описано принцип роботи системи, яка його використовує. Вказано коли і чому слід його використовувати.

В підрозділі 3.6 було описано випадки, в яких доцільно використовувати тип веб-застосунків WebAPI. Перераховано переваги та додаткові можливості, які забезпечує WebAPI. Також розглянуто засіб автоматичної генерації документації Swagger та вказано його переваги.

Найбільш популярний стандарт авторизації користувачів, який використовувався в розроблюваній системі, описаний в підрозділі 3.7. Вказано алгоритм авторизації за OAuth 2.0. Також розказано, що таке JWT, як він формується та як його можна використовувати в обраному способі авторизації.

В підрозділах 3.8 та 3.9 проведено огляд технологій, які були використані для створення клієнтської частини системи. Описано веб-технології HTML і CSS та принципи їх застосування. Зазначено основні відомості та особливості мови програмування JavaScript. А також проведено огляд JavaScript бібліотеки – React. Розказано принцип її роботи та перелічено основні переваги.

4 ОПИС ПРОЦЕСУ РОЗРОБКИ СИСТЕМИ

4.1 Реалізація клієнт-серверної архітектури

Обрана для створення системи клієнт-серверна архітектура означає, що система буде складатись з двох застосунків. Перший буде працювати на сервері, а другий на стороні клієнта. В серверній частині буде бізнес-логіка та взаємодія з базою даних. На клієнтській частині буде лише відображення необхідної інформації користувачу.

Серверна частина являє собою WebAPI застосунок, реалізований з використанням технології ASP.NET Core. Такий тип реалізації застосунку, насамперед, обраний через вимогу надати можливість використання системи для створення власних клієнтських застосунків користувача. Таке рішення стало також причиною використання інструмента для автоматичного створення документації. Створений WebAPI застосунок містить багато точок взаємодії які виконують всі необхідні дії з системою. Зазвичай такі кінцеві точки реалізовуються в контролерах. Будь який WebAPI контролер в ASP.NET Core є наслідником класу ControllerBase, який включає необхідну функціональність для обробки HTTP запитів. Якщо звернути увагу на ієрархію класів в ASP.NET Core, то можна побачити, що в застосунку створеного за шаблоном MVC користувацький контролер є наслідником класу Controller, який в свою чергу є наслідником класу ControllerBase. На відміну від цього користувацькі контролери у WebAPI є напряму наслідниками класу ControllerBase. Це зроблено тому, що клас Controller, що використовується в MVC, містить функціональність, яка актуальна тільки для MVC контролерів. Наприклад, метод View повертає розмітку сторінки. Проте це є зайвим для WebAPI контролера, оскільки він повинен повертати дані у відповідному форматі. Однак клас ControllerBase не має всієї необхідної функціональності, тому використо-

вують атрибут ApiController. Він дає можливість додавати фільтри, конвертувати дані, що прийшли в запиті у модель, яка очікується в якості параметра та перевіряти її стан.

Кожна кінцева точка у WebAPI має свій метод в контролері, який обробляє запит, що на нього прийшов. Такий метод називають action методом, приклад якого зображений на рисунку 4.1. Щоб вказати якого типу запит може обробляти даний action метод, слід вказати атрибут, в якому це зазначено. Існує кілька таких атрибутів: HttpGet, HttpPost, HttpPut, HttpDelete, HttpPatch. Кожен з них відповідає типу запита.

```
[HttpDelete("{id}")]
public async Task Delete(Guid id)
{
    await _commandBus.Execute(new DeleteClassCommand
    {
        Id = id
    });
}
```

Рисунок 4.1 – Action метод

В прикладі, наведеному на рисунку 4.1, зображено обробку запиту на видалення класу з системи. В нього присутній атрибут HttpDelete, що свідчить про метод запиту, який він обробляє. Також цей action метод містить параметр шляху. Він буде зв’язуватись з параметром id внутрішньою реалізацією ASP.NET Core. Оскільки це не GET, а DELETE запит, то відповідь залишається пустою.

Всі action методи описані в автоматично створеній документації з використанням Swagger. Для цього було підключено nuget пакет Swashbuckle.AspNetCore, який містить функціональність для конфігурування генерованої документації. Після цього потрібно додати Swagger до сервісів, які конфігуруються в класі Startup у методі ConfigureServices, та

використати Swagger в методі Configure скориставшись параметром типу IApplicationBuilder.

Клієнтську частина системи реалізовано з використанням JavaScript бібліотеки React. Для взаємодії з серверною частиною системи використовувався інтерфейс для виконання запитів FetchAPI. Він досить простий у використанні та його приклад можна побачити на рисунку 4.2. Щоб зробити запит слід викликати метод fetch, в який слід передати обов'язковий параметр – адресу куди робити запит. Другий параметр, в який передається об'єкт з даними про тип запиту, заголовком та тілом, є необов'язковий.

```
fetchSchools() {
  return fetch("http://localhost:8001/api/Schools", {
    method: "GET",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json"
    }
  })
    .then(response => response.json())
    .then(response => {
      this.setState({ schools: response.schools });
      window.localStorage.setItem(
        "schools",
        JSON.stringify(response.schools)
      );
    })
    .catch(reason => {
      console.error(reason);
    });
}
```

Рисунок 4.2 – Приклад використання FetchAPI

З наведеного прикладу видно, що для виконання HTTP запиту викликано метод fetch. В якості першого параметру йому передано адресу, на яку потрібно зробити запит. Другий параметр являє собою об'єкт,

який містить додаткову інформацію про запит. В ньому вказано тип запиту, який слід виконати. Також він містить інформацію про заголовок запиту. Заголовок складається з елементів типу ключ-значення. Результатом виконання буде Promise – об’єкт, який представляє собою результат успішного або неуспішного запиту та містить його результат. Для його обробки викликається метод then, який приймає делегат в якості параметра. Цей делегат виконується у випадку успішного виконання запиту. Він приймає відповідь на запит як параметр. Для обробки неуспішного запиту слід викликати метод catch. Він в якості параметра також приймає відповідь запиту.

Насправді будь-які результат виконання можна обробляти за допомогою метода then. Він обробляє різні типи станів Promise. Catch являє собою метод then, який обробляє лише відповіді в стані rejected.

4.2 Реалізація принципу CQRS

Для взаємодії з даними було використано підхід CQRS. Його реалізація почалась з виокремлення інтерфейсів, які будуть використовуватись для позначення типу запиту: ICommand, IQuery та IQueryResult. Для IQuery було додано параметр типу з обмеженням, що він повинен реалізовувати інтерфейс IQueryResult. Далі було виділено інтерфейси для обробників команд та запитів, приклад одного зображено на рисунку 4.3.

```
public interface ICommandHandler<TCommand>
    where TCommand : ICommand
{
    Task Execute(TCommand command);
}
```

Рисунок 4.3 – Інтерфейс обробника команд

Для інтерфейса ICommandHandler був доданий параметр типу, на який додано обмеження, що тип, переданий в якості параметра повинен реалізовувати інтерфейс ICommand. Також в ньомму було створено метод Execute, який повертає Task як результат, а як параметр приймає об'єкт, який має тип параметра типу. Для IQueryHandler було додано два параметри типу, на перший додано обмеження, що він повинен реалізовувати інтерфейс IQuery, а на другий, що від має реалізовувати інтерфейс IQueryResult. Було додано метод Execute, який повертає Task<> типу параметра типу, який реалізовує інтерфейс IQueryResult, а в якості параметра приймає тип параметра типу, що реалізовує інтерфейс IQuery. В інтерфейсах обробниках клас Task використовується для можливості виконувати ці методи асинхронно, але його можна опустити. Тоді обробники будуть виконуватись синхронно.

Щоб не створювати обробники необхідних команд або запитів щоразу було зроблено інтерфейси ICommandBus та IQueryBus для команд та запитів відповідно. В них буде лише один метод – Execute, який буде дізнаватись який обробник потрібно створити за допомогою типу команди або запити, що передано як параметр та створювати його за допомогою Dependency Injection (DI) контейнера. Для цього було використано nuget пакет Autofac. Перед тим як виконувати обробку запит або команда буде проходити перевірку на правильність. Для цього було використано бібліотеку FluentValidation.

Тобто кожна команда буде виконуватись в реалізації інтерфейсу ICommandBus, принцип роботи якого можна зрозуміти з додатку А. В ньому описано алгоритм роботи реалізації інтерфейсу ICommandBus та IQueryBus. Також приклад реалізації інтерфейсу ICommandBus зображено на рисунку 4.4.

Ці абстракції було винесено в окремий проект. І оскільки проектів в системі кілька, для уникнення дублювання коду, було створено власний

nuget пакет, який розміщений в локальному репозиторії пакетів. Він використовується лише в тих проектах, де потрібна така функціональність.

```
internal class CommandBus : ICommandBus
{
    private readonly ILifetimeScope _lifetimeScope;

    public CommandBus(ILifetimeScope lifetimeScope)
    {
        _lifetimeScope = lifetimeScope;
    }

    public async Task Execute<TCommand>(TCommand command) where TCommand : ICommand
    {
        var commandHandler = ResolveHandler<TCommand>(command);

        await Validate(command);

        await commandHandler.Execute(command);
    }
```

Рисунок 4.4 – Реалізація інтерфейсу ICommandBus

Після підключення вищезгаданого пакету було створено реалізації команд, запитів, їх обробників та валідаторів.

В результаті принцип роботи системи з підходом CQRS наступний. В action методі в контролері отримуємо команду або запит. В контролері є об’єкт IQueryBus та ICommandBus, який ініціалізується в конструкторі контролера за допомогою DI контейнера. Викликаємо у відповідного об’єкта метод Execute, в який передаємо команду або запит, який ми отримали. Після цього відбувається перевірка наявності обробника в DI контейнері, якщо він відсутній, викидаємо відповідне виключення. Якщо обробник визначено, валідуємо модель, використовуючи валідатор для отриманої команди або запиту. Якщо перевірка на валідність пройдена успішно викликаємо метод Execute в обробника. В цьому методі відбува-

ється безпосередня взаємодія з базою даних за допомогою Entity Framework.

4.3 Опис структури бази даних та сутностей

В розроблюваній системі існує 9 сутностей, які зображені в додатку Б на діаграмі зв’язків між сутностями. Ці сутності контролюються відповідними користувачами в системі.

Першою сутністю, з якої починається процес використання системи школою, є School. Вона являє собою сутність школи. В ній зберігаються наступні дані про навчальний заклад:

- Ідентифікатор;
- Назва школи;
- Ідентифікатор міста, в якому знаходиться;
- Логін;
- Пароль;
- Refresh token.

Ідентифікатор є унікальним для кожної школи. Назва та місто використовуються для опису школи. Між містом та школою наявний зв’язок один до багатьох. Він реалізований шляхом наявності в сутності школи ідентифікатора міста в якому вона знаходиться. Логін, пароль та refresh token використовуються для авторизації адміністратора школи, який створює всіх учасників системи в рамках школи.

Сутність City являє собою місто. Вона досить лаконічна і містить лише кілька властивостей:

- Ідентифікатор;
- Назва.

Ідентифікатор є унікальним для кожного міста та використовується для ідентифікації запису серед інших і створення посилань на відповідне місто. Назва відповідає назві міста.

Сутність Teacher описує користувача типу вчитель. Це кінцевий користувач системи, який безпосередньо бере участь в навчальному процесі. Він має наступні властивості:

- Ідентифікатор;
- Ім’я;
- Прізвище;
- Ім’я по-батькові;
- Логін;
- Пароль;
- Refresh token;
- Ідентифікатор школи, до якої належить.

Ідентифікатор, унікальний для кожного вчителя, використовується для посилання на відповідного користувача. Ім’я, прізвище та ім’я по-батькові використовуються для опису. Логін, пароль та refresh token використовуються для авторизації користувача в системі. Між школою та вчителем існує зв’язок один до багатьох. Він реалізовується шляхом додавання до сутності вчителя ідентифікатора школи, до якої він відноситься.

Сутність Class описує клас в системі та складається з наступних властивостей:

- Ідентифікатора;
- Назви;
- Ідентифікатора школи, до якої належить.

Унікальний ідентифікатор генерується для кожної школи при її створенні. Назва використовується для відображення зрозумілого корис-

тувачам імені класу. Також воно є унікальним на цілу школу. Ідентифікатор школи використовується для створення посилання на школу до якої він належить та формування зв’язку один до багатьох між школою та класом.

Сутність Pupil використовується для відображення в системі користувачів типу учень. Вона містить наступні властивості:

- Ідентифікатор;
- Ім’я;
- Прізвище;
- Логін;
- Пароль;
- Refresh token;
- Ідентифікатор класу, в якому навчається.

Ідентифікатор унікальний для кожного учня. Ім’я та прізвище використовуються для зрозумілого користувачу відображення конкретного учня. Логін, пароль та refresh token використовуються для авторизації користувача в системі. Оскільки між класом та учнем існує зв’язок один до багатьох, тому в учні є ідентифікатор класу, до якого він належить.

Сутність Discipline описує дисципліну в школі. Вона створена для забезпечення можливості школі створювати власний перелік дисциплін. Наступні властивості наявні в ній:

- Ідентифікатор;
- Назва;
- Ідентифікатор школи, до якої вона належить.

Ідентифікатор кожної дисципліни унікальний. Назва дисципліни також унікальна для школи, в якій вона знаходиться та використовується для зрозумілого користувачу відображення в системі. Ідентифікатор

школи використовується для забезпечення зв’язку один до багатьох між школою та дисципліною.

Сутність Subject описує предмет для конкретного класу з конкретним вчителем та дисципліною. В системі вона має наступні властивості:

- Ідентифікатор;
- Ідентифікатор класу;
- Ідентифікатор вчителя;
- Ідентифікатор дисципліни.

Ідентифікатор предмету унікальний в системі. Ідентифікатор класу використовується для створення зв’язку між класом та предметом один до багатьох. Для реалізації зв’язку між предметом та вчителем багато до одного додано властивість ідентифікатора вчителя. Ідентифікатор дисципліни використовується для реалізації зв’язку один до багатьох між дисципліною та предметом. Також слід зауважити, що комбінація ідентифікаторів вчителя, класу та дисципліни в сутності предмету є унікальна. Адже, якщо ця комбінація буде однакова для кількох записів, то це буде означати, що це один й той самий предмет. А один предмет повинен мати один запис в базі даних.

Сутність DayOfTheWeek створена для представлення дня тижня в системі. Хоча дні тижня є константами і є відповідний перелік в .NET, було прийнято рішення винести їх в базу даних. Так було зроблено, оскільки екземпляри цієї сутності будуть використовуватись у всій системі. І щоб не використовувати літерали або уникнути дублювання однакових переліків в різних програмах, створено єдине джерело днів тижня в системі. Ця сутність містить наступні властивості:

- Ідентифікатор;
- Назва дня тижня.

Сутність ScheduleCell використовується для описання комірки в розкладі. Ця комірка не прив’язана до конкретної школи, а є загальною для всіх. Тобто перший урок в школі А та в школі Б будуть мати посилання на одну й ту ж саму комірку. Вона містить наступні властивості:

– Ідентифікатор;

– Ідентифікатор дня тижня;

– Порядковий номер уроку;

Ідентифікатор комірки розкладу є унікальним в системі. Ідентифікатор дня тижня та порядковий номер вказують на час проведення уроку, за допомогою яких можна уявити яка саме комірка розкладу представлена.

Сутність Lesson характеризує урок, інформація про урок, який буде відображатись в розкладі. Вона має наступні властивості:

– Ідентифікатор;

– Номер аудиторії;

– Ідентифікатор предмета;

– Ідентифікатор уроку.

Ідентифікатор унікальний для кожного уроку. Номер аудиторії використовується для вказання інформації про місце проведення уроку. Для реалізації зв’язку один до багатьох між предметом та уроком створено ідентифікатор предмета. Для створення зв’язку між уроком та коміркою розкладу багато до одного використано властивість ідентифікатор комірки розкладу.

З діаграми зв’язків між сутностями, зображеної в додатку Б, видно що сутність Lesson має ідетифікатор предмета та ідентифікатор комірки розкладу, тобто вона використовується як проміжна таблиця між предметом та коміркою розкладу. За допомогою неї між цими сутностями при-

сутній зв’язок багато до багатьох. Цей зв’язок є основним для отримання розкладу користувача.

4.4 Створення авторизації в системі

В підрозділі 3.7 була описано принцип авторизації користувачів за стандартом OAuth 2.0. Його було дотримано в розробленій системі. Для цього було створено action метод для авторизації користувачів. Оскільки в системі 2 типи користувачів, то action методи для авторизації були створені окремо для учня та вчителя. На вході потрібно передати в нього логін та пароль користувача, який робить спробу авторизуватись. Слід зауважити, що зберігати паролі користувачів у відкритому доступі не надійно, тому після його отримання він хешується за алгоритмом SHA-512. Далі користувач відповідного типу з такими даними отримується з бази даних. Якщо користувача не знайдено, тобто вхідні дані не правильні, відповідне повідомлення повертається йому. У випадку якщо дані вірні та користувача знайдено, потрібні ідентифікатор користувача обирається для запису в access token.

В якості access token’а було використано JWT. В розробленій системі алгоритм створення JWT винесено в nuget пакет, оскільки ця функціональність необхідна в кількох проектах системи. Для її реалізації використано бібліотеку зазначену в [9] jose-jwt. Після створення JWT, він повертається у відповідь на запит авторизації як access token. Але оскільки access token має термін придатності, повинна бути можливість його оновлювати. З цією метою у відповіді на запит повертається refresh token.

Refresh token – випадково згенерований рядок сталої довжини. Вона генерується після отримання access token та зберігається в базу даних відповідному користувачу.

На клієнті ці 2 token’и повинні бути збережені для використання в авторизованих запитах. Для цього, при отриманні успішної відповіді на запит авторизації, token’и зберігаються в локальне сховище. За допомогою Swagger’а можна дізнатись, чи потрібна авторизація користувача для використання певної кінцевої точки. І якщо так, то access token додається в заголовок запиту. У випадку, якщо його термін придатності вичерпався, у відповідь приходить відповідне повідомлення з кодом помилки. Код помилки порівнюється з кодом помилки, який свідчить про те, що access token не правильний. Якщо це так, то робимо запит за новим access token’ом використовуючи refresh token.

На сервері система намагається знайти користувача з отриманим refresh token’ом. Якщо такого користувача знайдено, використовує його ідентифікатор для створення нового access token’а і повертає його у відповідь на запит.

Щоб уникнути дублювання коду, для перевірки access token’а на серверній частині системи було створено власний фільтр обробки HTTP запитів. Принцип його роботи полягає в тому, що при отриманні запиту на кінцеву точку, яка доступна лише для авторизованих користувачів, фільтр намагається отримати access token із заголовка. Після цього він перевіряє чи не вичерпався термін придатності token’а. Наступним кроком з token'а дістається корисна частина, в якій є ідентифікатор користувача. За допомогою ідентифікатора шукаємо користувача в базі даних та після його отримання присвоюємо його полю User абстрактного контролера AuthorizedController. Це поле буде містити авторизованого користувача, що дасть можливість використовувати необхідні дані про користувача, коли це буде необхідно. Якщо будь-яка з дій при отриманні інформації про користувача за допомогою ідентифікатора була неуспішна, помилка з відповідним повідомленням та кодом повернеться у відповіді на запит.

4.5 Структура проекту клієнтської частини

Проект клієнтської частини було створено за допомогою інструмента create-react-app. Він створив базову структуру проекта з кількома файлами, які не використовувались в системі, тому вони були видалені.

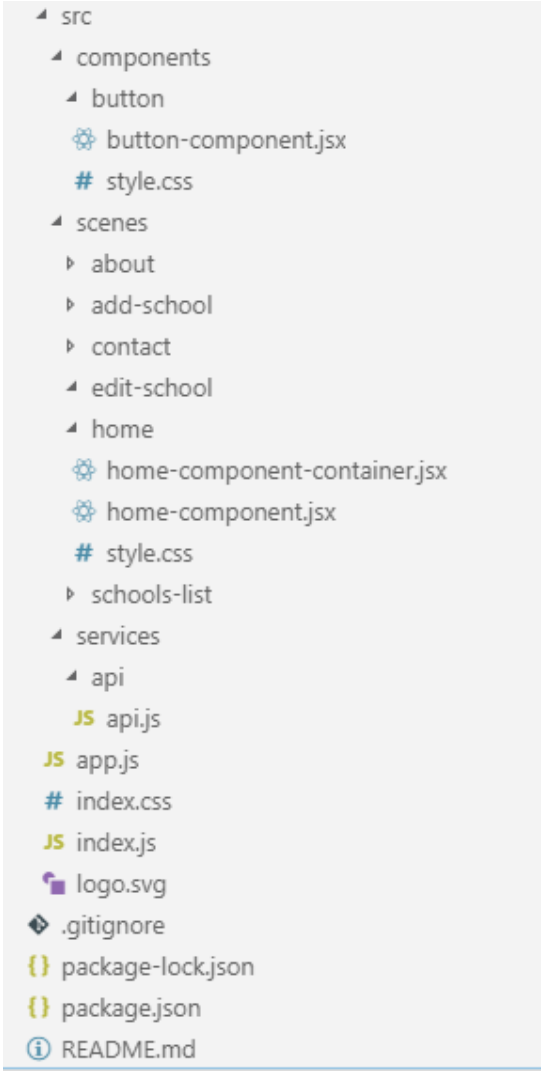


Рисунок 4.5 – Структура проекту клієнтської частини

На рисунку 4.5 зображено структуру проекту, отриману в результаті всіх змін з [10]. В ній на верхньому рівні є 2 папки public та src. В папці public зберігаються html та css які відображаються користувачу при вході в систему.

Назва папки обумовлена тим, що код файлів в ній доступний в інструментах розробника в браузері. Щодо директорії src , то зазвичай файли, що знаходяться в ній мініфікуються і компілюються в кінцевий вигляд. Тому вони не є публічними. Саме в цій директорії є всі компоненти та інші файли з вихідним кодом.

В папці src знаходиться файл index.js в якому у потрібному блоці створюється react компонент App. Він описується у файлі app.js на тому ж рівні. В ньому було вказано правила роутингу в застосунку. Для цього було використано бібліотеку react-router. Також на цьому рівні є файл з базовими стилями. Окрім цього тут знаходяться ще 3 директорії.

В директорії components знаходяться базові компоненти, які часто використовуються в системі. Наприклад, компонент button, який використовується для відображення кнопки в системі. Його розмітка знаходиться у файлі button-component.jsx, а на одному рівні з ним знаходяться файл зі стилями для кнопки – style.css.

В директорії scenes знаходяться компоненти сторінок системи. Для кожної сторінки створюється власна директорія, в якій є файл з компонентом, файл з контейнером для компонента та файл із стилями. Наприклад, для домашньої сторінки в директорії home існує її компонент, який відповідає за відображення. Також в ній є файл зі стилями, який підключається в home-component.jsx. І ще в цій директорії присутній контейнер home-component-container.jsx. Основна мета контейнера – розділити відображення компонента від логіки та взаємодії з серверною частиною. Для цього код, який не відповідає за відображення знаходиться в контейнері і виконується першим. Після того, як всі дані для відображення готові контейнер викликає звичайний компонент, передаючи йому всі необхідні дані в якості властивостей. Використовувати scenes в інших компонентах не правильно. Також в цій директорії присутній файл стилів style.css, який використовується в компоненті home-compoent.jsx.

В директорії services знаходяться допоміжні сервіси. Наприклад, тут знаходиться сервіс, який відокремлює логіку взаємодії з серверною частиною – api.js. Ці сервіси можуть використовуватись в будь-яких компонентах.

4.6 Висновки до розділу 4

В розділі 4 розглянуто спосіб реалізації розробленої системи. В підрозділі 4.1 більш детально розглянуто обрану архітектуру системи. Заначено деталі її реалізації з використанням обраного набору технологій. Використаний підхід CQRS детально описано в підрозділі 4.2. В ньому зазначено основні складові цього підходу в розробленій системі. В підрозділі 4.3 описано структуру бази даних. Там вказано які сутності існують в системі, які зв’язки між ними, як вони реалізовувались та детально описано властивості кожної сутності і для чого кожна з них створювалась. В розділі 4.4 детально розказано про процес авторизації користувача в системі. Описано стандарт, якого було дотримано в ході реалізації та підхід який був використаний. В підрозділі 4.5 розказано про структуру реалізованого клієнтського веб-додатку. Розписано про використаний принцип структурування проекту та аргументовано використання всіх частин в створеній структурі.

5 КЕРІВНИЦТВО ВИКОРИСТАННЯ СИСТЕМИ

5.1 Принцип роботи системи

Принцип роботи системи формувався на основі існуючих рішень оглянутих в підпункті 1.3. За ним вирішено створити кілька застосунків, доступ до яких буде надаватись відповідним користувачам. А саме, застосунок для адміністраторів шкіл, який називається School System Management, та застосунок для учнів та вчителів, який називається School System. Окрім цього буде ще один окремий застосунок для створення облікових записів адміністраторам шкіл. Він називається School System Administration. Діаграма прецедентів знаходиться в додатку В.

У випадку, якщо школа хоче використовувати розроблену систему, її представник зв’язується з представником системи через мобільний або надіславши лист на електронний адрес, який вказаний на сайті системи. Після цього працівник школи повинен повідомити обраним способом інформацію про школу, яку він намагається підключити до системи. Використовуючи ці дані працівник системи створює новий обліковий запис школи в системі та надає логін та пароль до нього представнику школи. Зробити це можна за допомогою застосунку School System Administration. Саме в ньому є можливість створення редагування, видалення та перегляду існуючих шкіл.

Після цього представник школи передає отримані дані людині, яка буде адмініструвати цю школу. Це може бути як той хто контактував від імені школи, так і зовсім інша людина. За допомогою них обраний адміністратор школи зможе авторизуватись в обліковому записі школи в застосунку School System Management. Тут він може створити вчителів та учнів, які будуть кінцевими користувачами системи. Також в цьому застосунку адміністратор може сформувати класи, створити перелік дисци-

плін для школи та додати розклад для будь-якого класу. Далі адміністратору школи потрібно поширити дані необхідні для авторизації, логін та пароль, з вчителями та учнями. Після цього в застосунку School System Management адміністратор зможе переглядати статистику про школу, а саме кількість учнів в школі, середній бал всіх учнів в школі. Це допоможе школі аналізувати продуктивність своєї роботи.

Основне користування системою відбувається в застосунку School System. Він створений для кінцевих користувачів – вчителів та учнів. Їхню роботу він і покликаний пришвидшити шляхом автоматизації більшості необхідної функціональності. Тут користувач після авторизації в своєму обліковому записі потрапляє на сторінку з розкладом уроків.

Якщо користувач – учень, він бачить розклад уроків для свого класу. В розкладі відображається інформація про назву уроку, час та місце його проведення. Також учень має змогу перейти на сторінку предмету. Там він побачить інформацію про вчителя, який викладає предмет, всі дні та час, коли проводиться урок. Інформація про домашнє завдання за весь час теж можна побачити на цій сторінці. Окрім розкладу учню також доступний журнал. В ньому він має змогу переглянути всі свої оцінки з будь- якого предмету. Оцінки інших учнів йому не доступні. Учень має змогу переглянути інформацію про себе в своєму профілі. Окрім загальної інформації там він бачить статистичну інформацію, наприклад, середній бал, яка допомагає йому зрозуміти рівень його знань. Учень має можливість переглянути інформацію про свій клас, де побачить список однокласників, середній бал класу.

Якщо в якості користувача є вчитель, то після авторизації в свій обліковий запис він бачить свій розклад уроків. В кожному уроці йому відображаються дані про клас у якого він веде його, а також час та місце проведення. Тут в нього є можливість перейти на сторінку детальної інформації про предмет. На ній буде інформація про клас в якого цей урок

відбудеться, всі дні та час коли він проводиться та список домашнього завдання по цьому предмету за весь час. Окрім цього вчитель може додати нове домашнє завдання або редагувати останнє. Вчитель має змогу перейти на сторінку класу, де побачить список всіх його учнів. Тут він зможе перейти на сторінку конкретного учня, щоб переглянути детальну інформацію про нього.

5.2 Процес користування системою

Для відображення процесу використання створеної системи використано застосунок School System Administration. В ньому працівник системи має змогу створювати облікові записи шкіл, якими будуть користуватись адміністратори шкіл для створення користувачів, дисциплін та розкладу уроків в конкретній школі.

Користування системою починається з сторінки, де працівник бачить список всі існуючих шкіл, зображеної на рисунку 5.1.

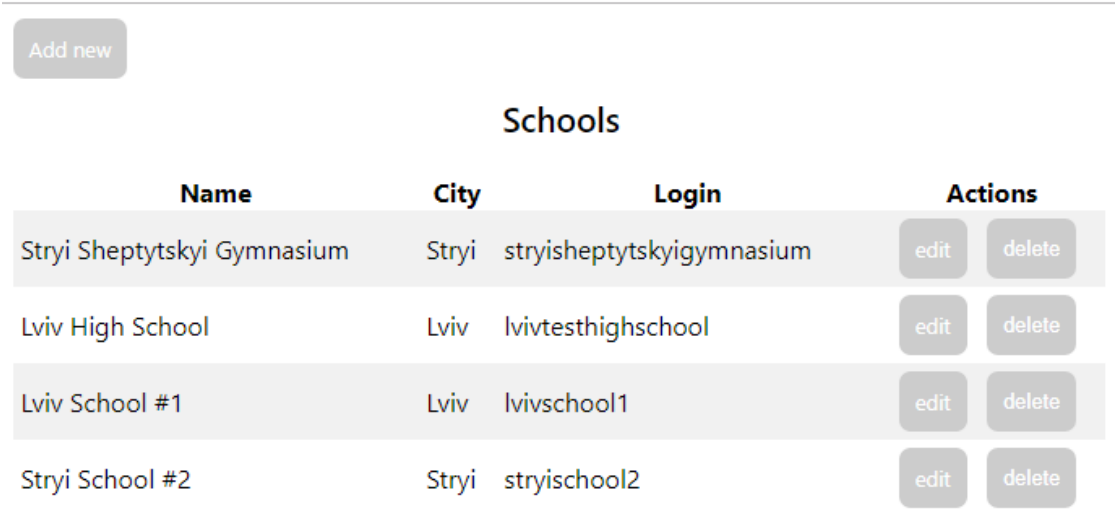


Рисунок 5.1 – Сторінка зі списком всіх створених шкіл

На ній працівник системи бачить основну інформацію про кожну з існуючих шкіл, а саме назву, місто та логін. Також тут в нього є можливість видалити обліковий запис школи. Для цього слід навпроти бажаної школи натиснути кнопку «delete», після чого таблиця зі школами оновиться.

З цієї сторінки користувач має змогу перейти на сторінку створення нової школи або редагування існуючої. Щоб додати новий обліковий запис школи, йому слід натиснути на кнопку «Add new». Це відобразить користувачу сторінку з формою для створення облікового запису школи. Її приклад зображена на рисунку 5.2.

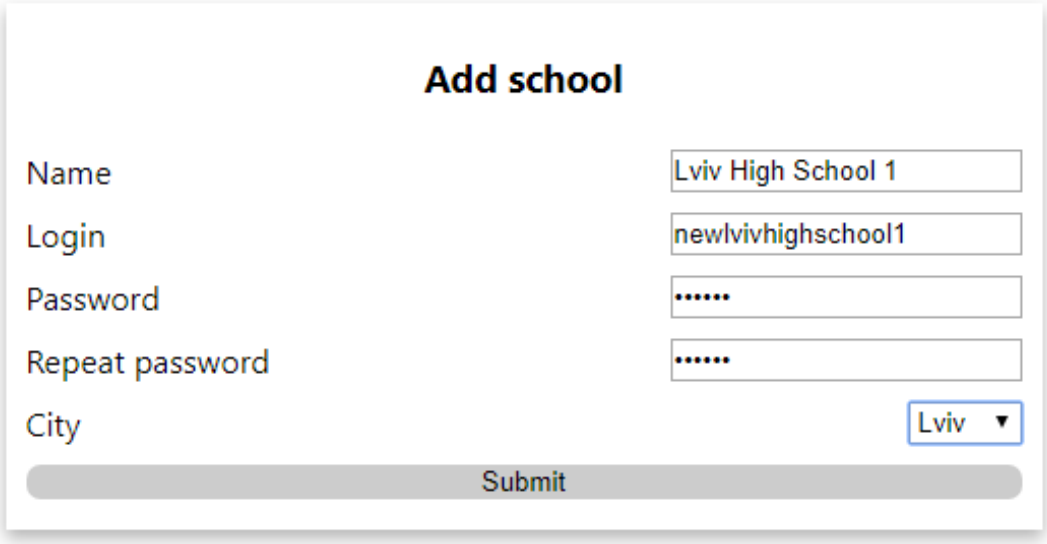


Рисунок 5.2 – Форма створення школи

На ній користувачу слід ввести назву нової школи, логін та пароль, а також йому необхідно обрати місто з переліку існуючих. Введений пароль слід повторити. Це зроблено для того, щоб впевнитись що користувач не помилився при введені паролю перший раз. Після цього користувач повинен натиснути кнопку «Submit». Якщо всі дані правильні, то створиться новий обліковий запис школи, а користувача переведе на сторінку зі списком усіх облікових записів шкіл. Якщо певні дані будуть

введені не правильно, внизу форми користувачу відобразиться повідомлення, в якому вказано що саме призвело до помилки.

Для редагування існуючого облікового запису школи користувач повинен натиснути кнопку «edit» навпроти відповідного облікового запису школи. Після цього його переведе на сторінку з формою редагування облікового запису школи, зображену на рисунку 5.3, поля в якій при завантажені сторінки будуть заповнені даними про обраний обліковий запис школу.

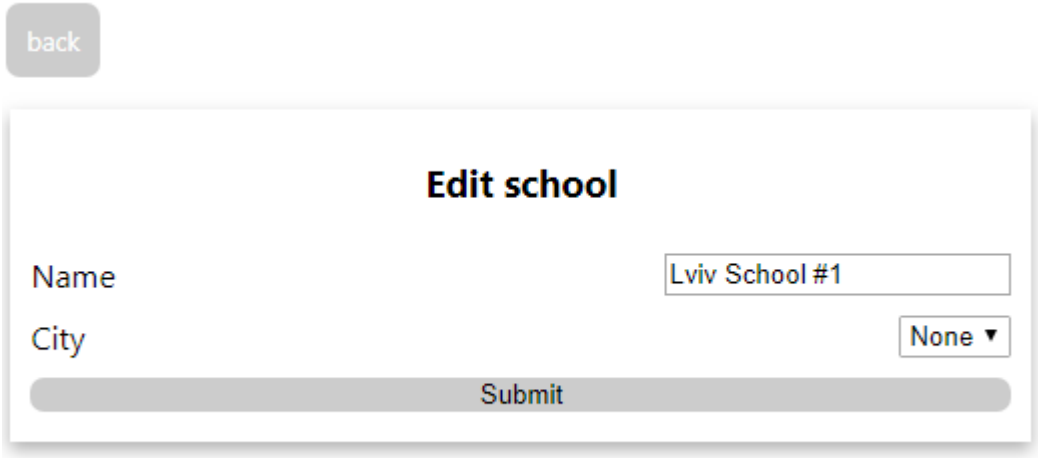


Рисунок 5.3 – Форма редагування школи

На ній користувач має змогу редагувати дані облікового запису школи, а саме змінити назву школи та місто в якому вона знаходиться. На цій формі також присутня перевірка на правильність введених даних. Тобто у випадку некоректно введеної інформації система не дасть зберегти зміни, а відобразить на формі повідомлення про помилку. В ньому буде вказано, які саме дані користувач ввів не правильно. Після цього користувача буде повернено наформу редагування школи, щоб він зміг швидко виправити невірно вказані дані.

5.3 Керівництво розробнику по інтеграції власного клієнтського застосунку з системою

Однією з особливостей системи є можливість створювати на її основі власні клієнтські застосунки. Для цього користувачам, які захотять нею скористатись надано відкритий API. Він описаний за допомогою інструмента автоматичного генерування документації Swagger. Відкритий інтерфейс взаємодії є для застосунків School System Management, який використовується адміністраторами шкіл, та School System, створений для кінцевих користувачів – учнів та вчителів.

5.3.1 Опис доступного API та документації до нього

В кожному з вищезгаданих інтерфейсів взаємодії перечисллено доступні кінцеві точки системи через які можна взаємодіяти з нею. Кожна така точка приймає HTTP запити та описується наступною інформацією:

- адреса, на яку потрібно робити запит;
- приклад вхідних параметрів, які очікуються системою в цьому запиті;
- модель успішної відповіді системи;
- модель неуспішної відповіді системи.

Приклад однієї з таких точок, зображений на рисунку 5.4, використовується для отримання списку міст, які є в системі.

З його опису можна побачити, що це GET запит. Шлях, на який слід робити запит для отримання даних, зображений в його заголовку. Нижче зображено приклад вхідних даних та їхнє знаходження в запиті або, як в прикладі, вказано про їх відсутність.

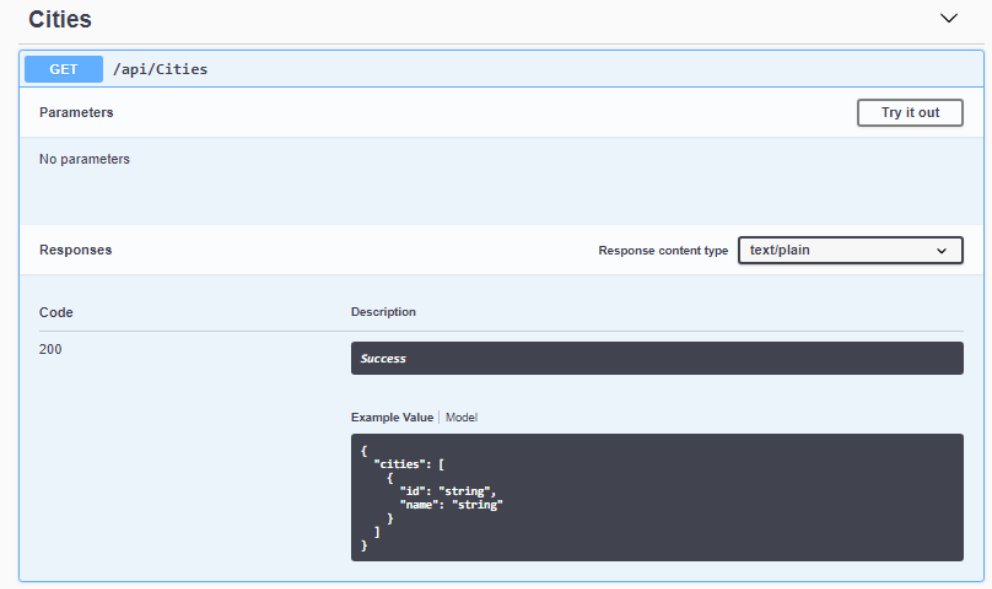


Рисунок 5.4 – Приклад опису точки взаємодії

Під запитом описано відповідь системи. Вказано тип контенту відповіді, та модель, яка повернеться в разі успішного запиту. Модель описується статус-кодом, який свідчить про її успішність, та тіло відповіді. В наведеному прикладі обрано тип контенту відповіді text/plain. На успішну спробу авторизації користувачу повернеться статус-код 200 та тіло, яке являтиме собою JSON об'єкт зі списком шкіл всередині. З прикладу моделі можна зрозуміти тип даних кожної властивості, яка є у відповіді. В даному випадку cities – масив JSON об’єктів, кожен з яких містить id та name типу рядок.

5.3.2 Процес авторизації в системі

На більшості кінцевих точках потрібно знати від імені якого користувача зроблено запит. Тобто слід виконати авторизований запит, приклад якого наведено в додатку Г. Для цього використано процес авторизації в системі у створеному інтерфейсі взаємодії. Створено відповідні кінцеві точки, які повертають необхідну інформацію про користувача для

його ідентифікації. Приклад однієї з них зображено на рисунку 5.5. Після отримання цієї інформації її слід використати у відповідних кінцевих точках.

Оскільки всі запити для авторизації користувачів мають однакові вхідні параметри та моделі відповідей і відрізняються лише адресом, на який робити запит, то розглянуто лише один з них – запит авторизації учня в системі.

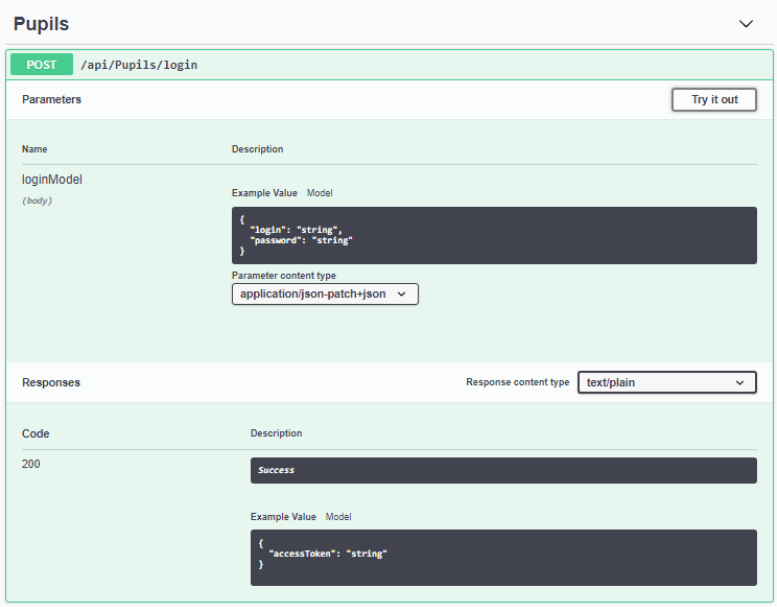


Рисунок 5.5 – Опис запиту на авторизацію користувача в системі

З його опису можна побачити, що це POST запит. Шлях, на який слід робити запит зображений в його заголовку. Нижче зображено приклад вхідних даних та їхнє знаходження в запиті. В даному прикладі вхідними даними є JSON об’єкт, який в собі містить властивості логін та пароль користувача, що авторизується. Зліва від прикладу вхідних даних вказано, що ці дані повинні бути записані в тіло запиту. Під прикладом вхідних зазначено тип вхідних параметрів, який розміщують в заголовку запиту.

Під запитом описано відповідь системи. Вказано тип контенту відповіді, та модель, яка повернеться в разі успішного запиту. Модель описується статус-кодом, який свідчить про її успішність, та тіло відповіді. В наведеному прикладі обрано тип контенту відповіді text/plain. На успішну спробу авторизації користувачу повернеться статус-код 200 та тіло, яке являтиме собою JSON об'єкт з властивістю access token всередині. Тип цієї властивості – рядок.

Після отримання access token'а користувача можна від його імені виконувати дії в системі, які вимагають авторизації. Для цього необхідно отриманий access token додати в заголовок запиту з ключем «Authorization». Приклад запиту, який вимагає авторизації зображено на рисунку 5.6.

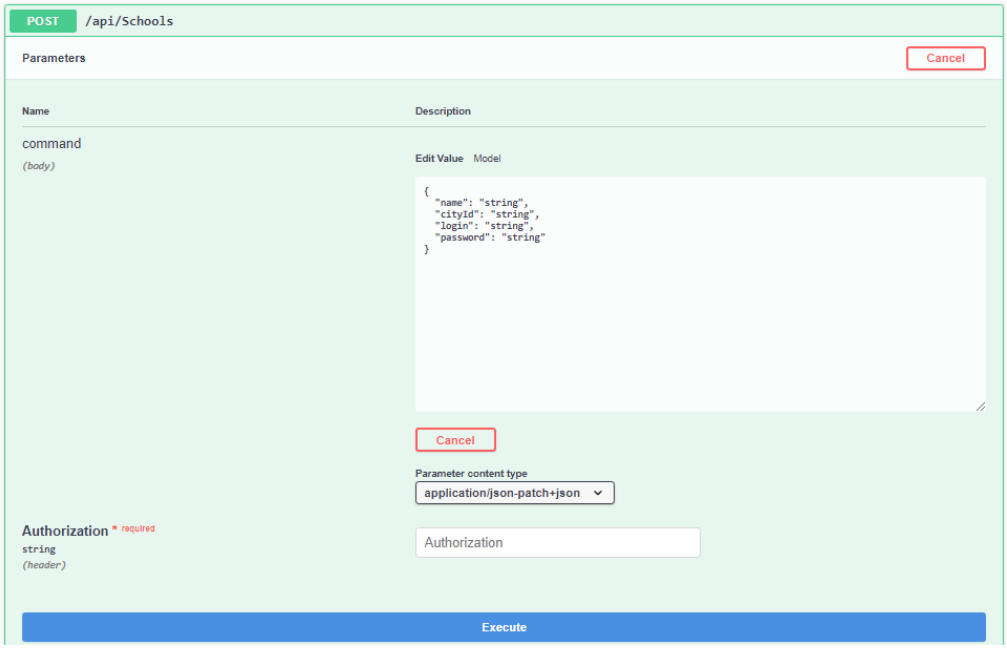


Рисунок 5.6 – Запит, який вимагає авторизацію

В документації створено відповідні поля для заголовків визначених в системі, які можна до них додати. Також зазначено чи вказаний заголовок обов'язковий чи ні.

5.3.3 Моделі помилок

У випадку неуспішного запиту користувачу у відповідь повертається об’єкт, який містить необхідну інформацію про помилку. За допомогою неї розробник, що виконує інтеграцію з системою, може виконувати обробку помилки відповідним чином. Слід зазначити, що коди помилок не будуть змінюватись, проте не потрібно їх використовувати для відображення кінцевим користувачам системи. Також повідомлення, яке містить опис помилки не повинно показуватись на користувацькому інтерфейсі. Адже воно може містити технічні деталі, які не будуть зрозумілі користувачу.


```
{
  "RequestId": "80000008-0002-fb00-b63f-84710c7967bb",
  "Content": {
    "InvalidObject": {
      "Name": "Lviv School #3",
      "CityId": "a87facff-4d6a-436b-b6b0-86ca419deffc",
      "Login": "lvivschool1",
      "Password": "SusgALneWFj15eC37aUvJTyvGVgsZ8u7RTvmmH7MG68n11Zv
    },
    "ValidationFailures": [
      {
        "PropertyName": "Login",
        "ErrorMessage": "School with this login already exist",
        "AttemptedValue": "lvivschool1"
      },
      {
        "PropertyName": "Name",
        "ErrorMessage": "School with this name already exist",
        "AttemptedValue": "Lviv School #3"
      }
    ],
    "Message": "Object in invalid state",
    "Code": "cqrs-0008"
  }
}
```


Рисунок 5.7 – Модель помилки

На рисунку 5.7 зображено модель помилки, яка може виникнути в процесі користування системи. Кожна помилка відображається у вигляді JSON об'єкта. Наступні властивості в ньому є обов’язкові:

– RequestId – ідентифікатор запиту, за допомогою якого в разі необхідності можна дізнатись додаткові дані про помилку;
– Message – повідомлення, яке описує зміст помилки;
– Code – код помилки використовується для ідентифікації типу помилки. Саме на цю властивість об’єкта відповіді слід орієнтуватись розробникам при обробці помилок в своїх застосунках. Вона ідентифікує тип помилки та завжди залишається незмінною.

В даному випадку наведено приклад помилки валідації вхідного запиту. Тобто передані дані, які були передані в запиті, не правильні. Окрім вищезазначених властивостей в об’єкті відповіді доступна ще одна властивість – Content. Вона відображається у випадку помилки валідації і містить в собі властивість типу об’єкт з масивом помилок в середині. В ньому вказано вхідні дані, які було надіслано, та причину помилки валідації. В даному випадку сказано, що школа з такою назвою вже існує. Тобто користувачу потрібно ввести іншу назву школи.

5.4 Висновки до розділу 5

В 5 розділі було розглянуто керівництво використання системи. Зазначено як розробникам інтегруватись з системою та як нею користуватись кінцевим користувачам.

В підрозділі 5.1 було розглянуто принцип використання системи. Система складається з трьох веб-застосунків: School System Administration – для створення облікових записів шкіл, School System Management – для керування розкладом, дисциплінами, вчителями та класами школи, School System – для моніторингу розкладу уроків.

В підрозділі 5.2 детально розглянуто процес користування системою. Наведено приклад використання застосунку School System. Описано всі дані, які доступні користувачу в ньому.

В підрозділі 5.3 описано керівництво розробнику по інтеграції власного клієнтського застосунку з системою. Вказано доступний API кінцеві точки, через які з ним можна взаємодіяти. Зазначено доступну документацію. Описано процес авторизації в системі, результатом якого є отримання access token’а і його використання для авторизації в системі. Також наведено моделі помилок системи. Всі вони приведені до однакового вигляду, щоб дати можливість розробнику, який інтегрується з системою, легко їх обробляти.

ВИСНОВКИ

Під час виконання дипломного проекту було розглянуто доцільність створеної системи, проведено порівняння з існуючими рішеннями виявлено їхні основні переваги та недоліки та на основі цього виокремлено список основної функціональності системи.

Сформовані вимоги до системи було поділено на технічні та функціональні. Про кожні з них було розписано більш детально, щоб отримати повне розуміння кінцевої системи.

Проаналізовані вимоги дали можливість обрати тип архітектури застосунку, підібрати підхід, який ліг в основу роботи системи, та визначитись з набором технологій, який було використано для її реалізації. В результаті створена система є ефективна та гнучка для внесення нової функціональності. Було використано клієнт-серверну архітектуру, яка була реалізована за допомогою ASP.NET Core та JavaScript бібліотеки React. На серверній частині для взаємодії з базою даних реалізовано підхід CQRS з використанням ORM системи Entity Framework. Дані зберігаються в базі даних, яка керується за допомогою СУБД MSSQL Server.

Проведено опис реалізації основних аспектів системи та рішень, прийнятих під час її створення. Детально розписано імплементацію клієнт-серверної архітектури. Наведено діаграму сутностей та описано кожну з них

Був детально розглянутий принцип роботи системи. Описано процес користування системи з необхідними прикладами. Також було оглянуто створену можливість інтегрування з системою. Наведено відповідні рисунки, які відображають описану інформацію.

В результаті створено систему, яка вирізняється своєю доступністю та відкритим інтерфейсом взаємодії, за допомогою якого можна створювати власні застосунки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Всеукраїнський освітній портал Класна Оцінка. [Електронний ресурс]: http://klasnaocinka.com.ua/
2. Електронний журнал | Електронний щоденник - online система для навчального процесу. [Електронний ресурс]: https://www.ukrschools.com.ua/
3. Щоденник.ua - шкільна освітня мережа. [Електронний ресурс]: http://shodennik.ua/
4. METANIT.COM - Сайт про програмування. [Електронний ресурс]: https://metanit.com/sql/sqlserver/1.1.php
5. Введення в CQRS + Event Sourcing: Частина1. Основи/ Хабр [Електронний ресурс]: https://habr.com/ru/post/146429/
6. П’ять простих кроків для розуміння JSON Web Tokens (JWT) / Хабр [Електронний ресурс]: https://habr.com/ru/post/340146/
7. Основи HTML - Вчимо веб-розробку | MDN [Електронний ресурс]: https://developer.mozilla.org/uk/docs/Learn/Getting_started_with_the_web/HTML_basics
8. CSS basics - Learn web development | MDN [Електронний ресурс]: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics
9. JSON Web Tokens - jwt.io [Електронний ресурс]: https://jwt.io/
10. How to better organize your React applications? – Alexis Mangin – Medium [Електроний ресурс]: https://medium.com/@alexmngn/how-to-better-organize-your-react-applications-2fd3ea1920f1
11. Materiály XV Mezinárodní vědecko - praktická konference «Vědecký pokrok na přelomu tysyachalety -2019», Volume 13 : Praha. Publishing House «Education and Science» -80 s

ДОДАТОК А

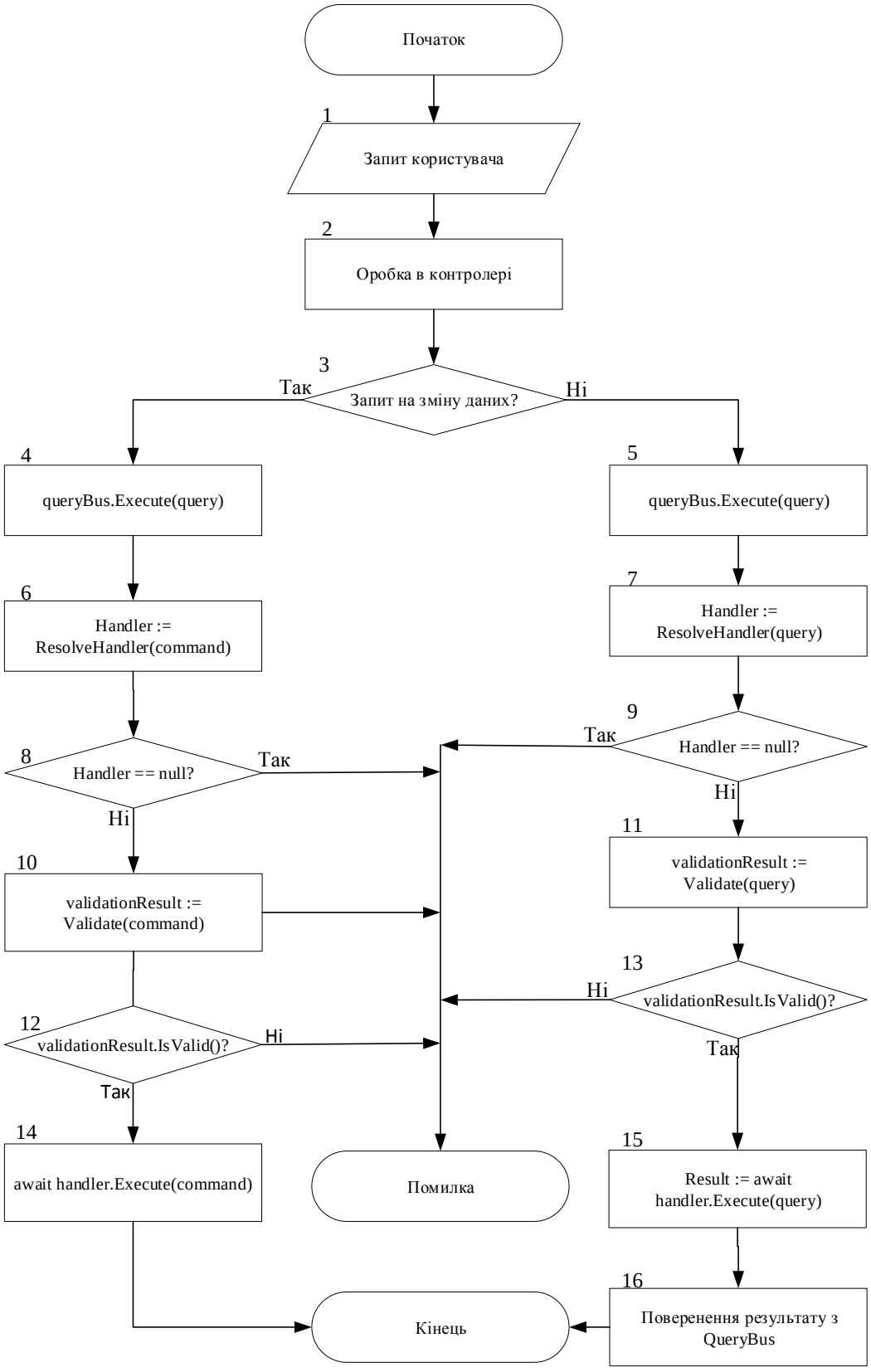


Рисунок А.1 – Блок-схема алгоритму обробки запитів за допомогою реалізацій інтерфейсів ICommandBus та IQueryBus

ДОДАТОК Б

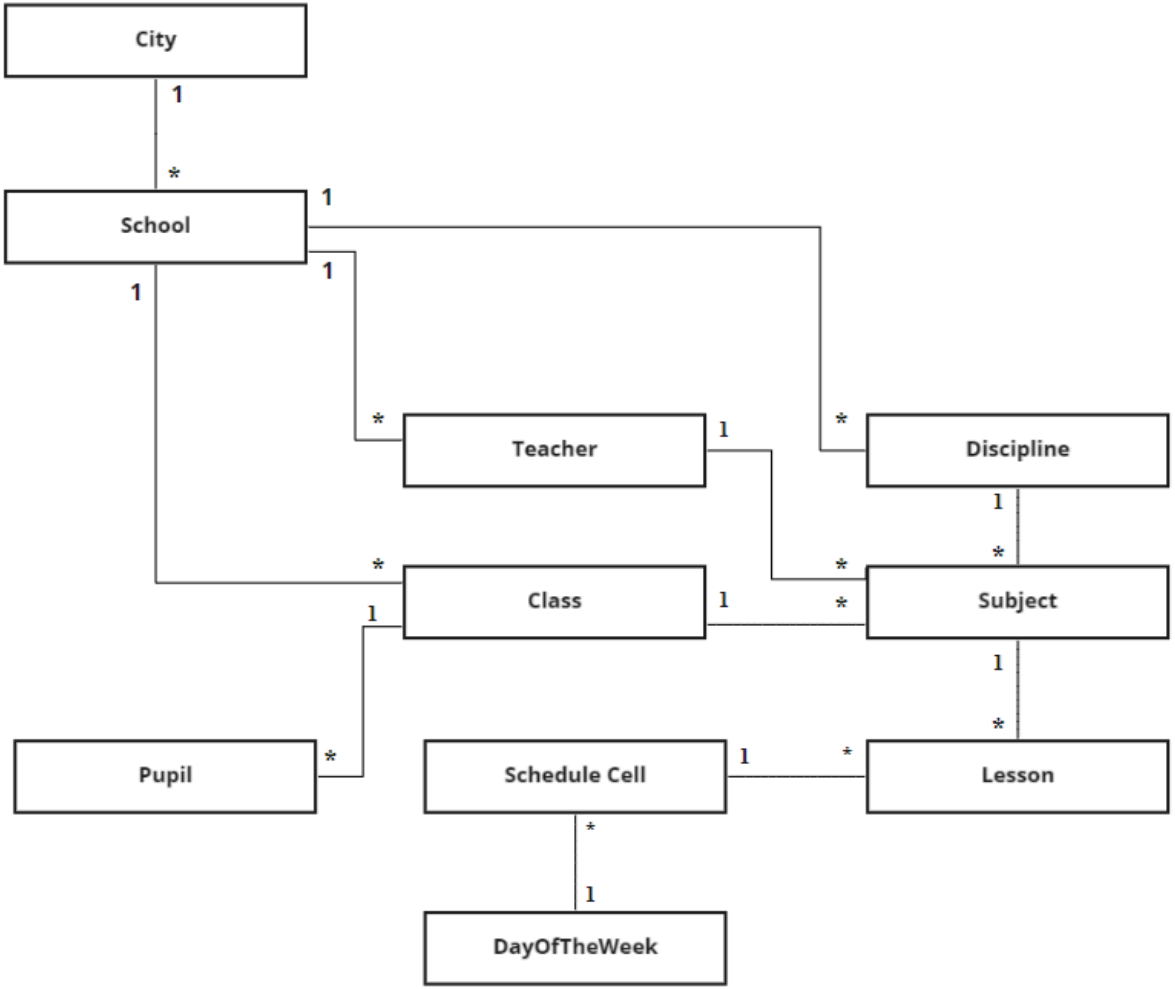
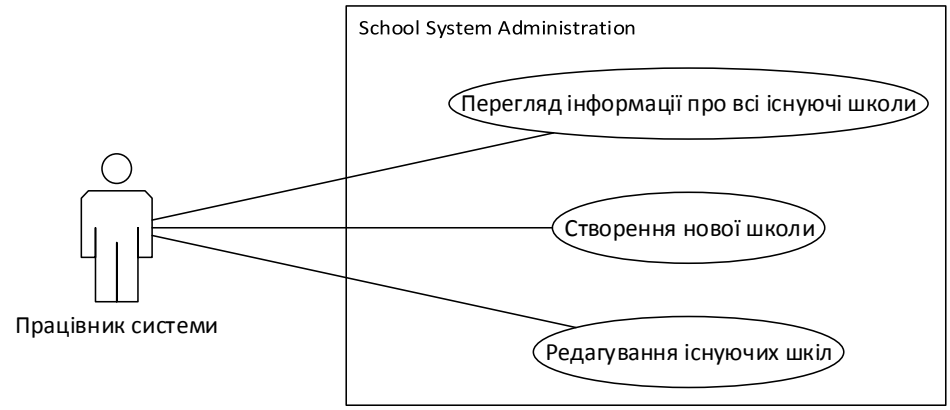
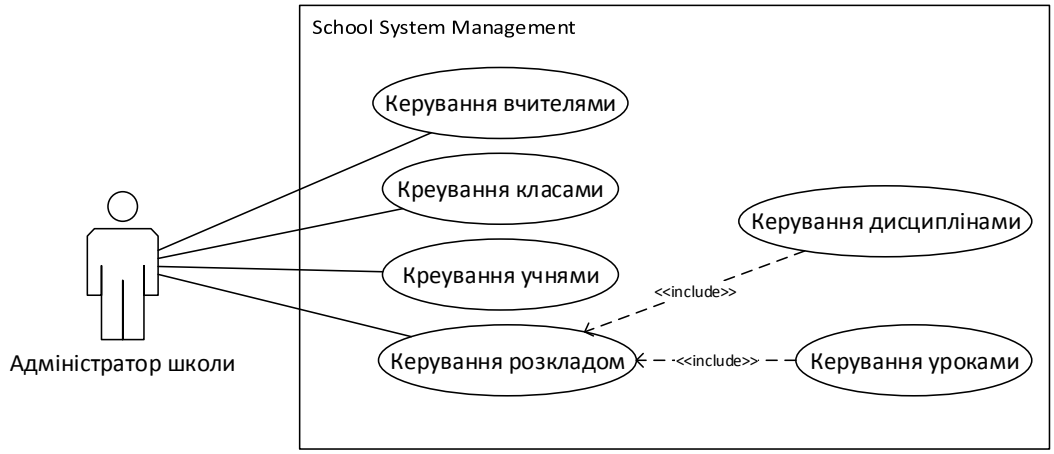


Рисунок Б.1 – Діаграма зв'язків між сутностями

ДОДАТОК В





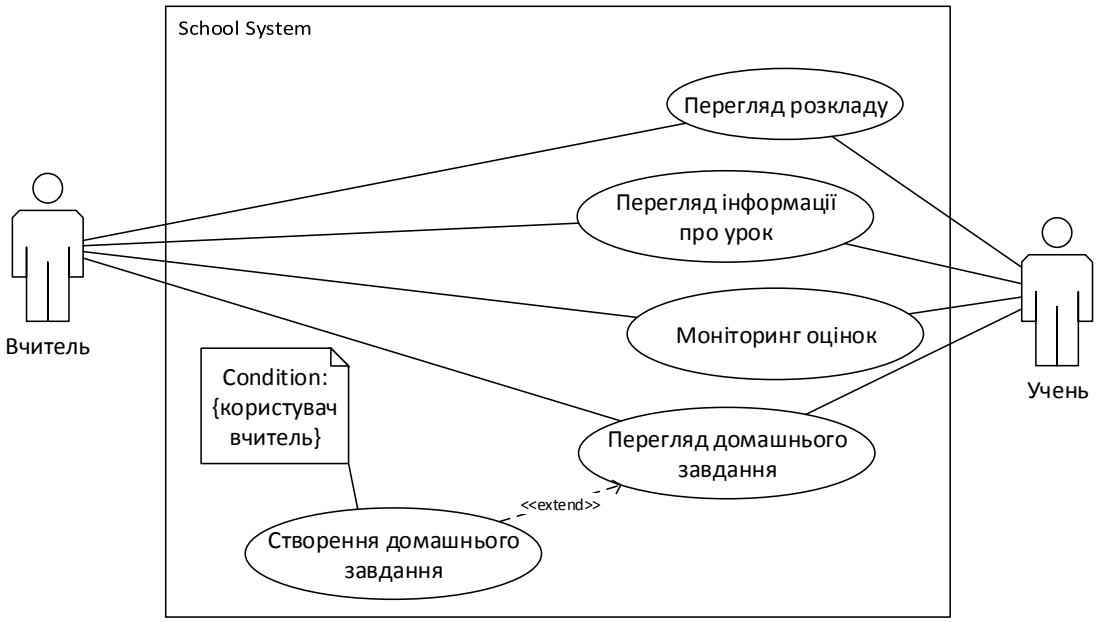


Рисунок В.1 – Use-case діаграма системи

ДОДАТОК Г

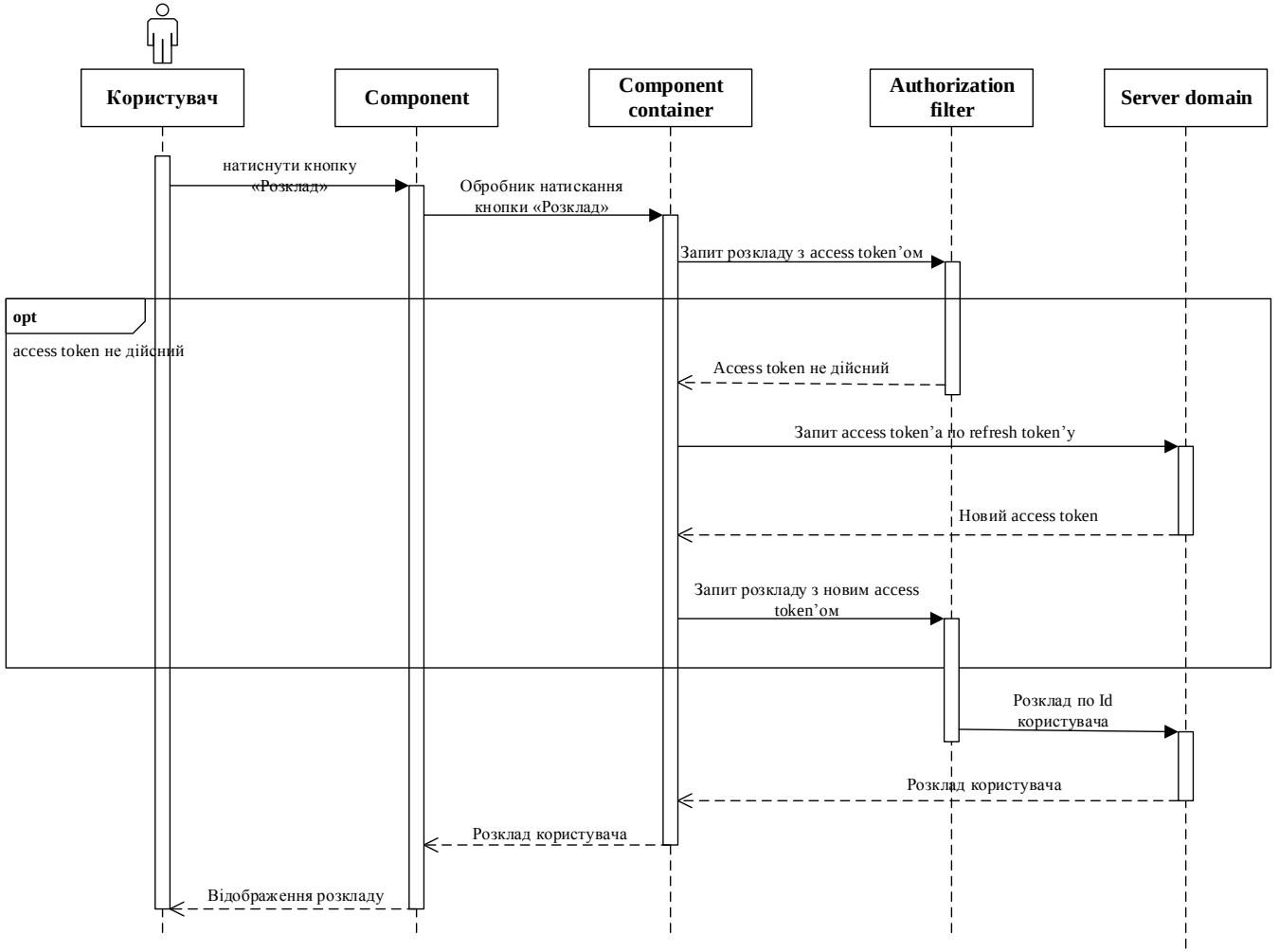


Рисунок Г.1 – Діаграма послідовності авторизованого запиту розкладу

					IT51.070БАК.006 Д4	
Ізм.	Лист	№ докум.	Підпис	Дата		

ДОДАТОК Д

Вихідний код

```
using Microsoft.EntityFrameworkCore;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context.Mo
delBuilders.Abstractions;

namespace
SchoolSystem.Management.Context
{
   public class
SchoolSystemManagementDbContext :
DbContext
   {
      private readonly
IModelBuilder<Class>
_classModelBuilder;
      private readonly
IModelBuilder<Teacher>
_teacherModelBuilder;
      private readonly
IModelBuilder<Pupil>
_pupilModelBuilder;
      private readonly
IModelBuilder<Discipline>
_disciplineModelBuilder;
      private readonly
IModelBuilder<Subject>
_subjectModelBuilder;

      public DbSet<Class> Classes { get;
set; }
      public DbSet<School> Schools {
get; set; }
      public DbSet<City> Cities { get;
set; }
      public DbSet<Teacher> Teachers {
get; set; }
      public DbSet<Pupil> Pupils { get;
set; }
      public DbSet<Discipline>
Disciplines { get; set; }
      public DbSet<Subject> Subjects {
get; set; }
      public DbSet<Lesson> Lessons {
get; set; }
      public DbSet<ScheduleCell>
ScheduleCells { get; set; }
      public DbSet<DayOfTheWeek>
DaysOfTheWeek { get; set; }

      public
SchoolSystemManagementDbContext(D
bContextOptions options,
IModelBuilder<Class>
classModelBuilder,
         IModelBuilder<Teacher>
teacherModelBuilder,
IModelBuilder<Pupil>
pupilModelBuilder,
         IModelBuilder<Discipline>
disciplineModelBuilder,
IModelBuilder<Subject>
subjectModelBuilder) : base(
         options)
      {
         _classModelBuilder =
classModelBuilder;
         _teacherModelBuilder =
teacherModelBuilder;
         _pupilModelBuilder =
pupilModelBuilder;
         _disciplineModelBuilder =
disciplineModelBuilder;
         _subjectModelBuilder =
subjectModelBuilder;
      }
```

```
        protected override void
OnModelCreating(ModelBuilder
modelBuilder)
        {

base.OnModelCreating(modelBuilder);

_classModelBuilder.Build(modelBuilder.
Entity<Class>());

_teacherModelBuilder.Build(modelBuild
er.Entity<Teacher>());

_pupilModelBuilder.Build(modelBuilder
.Entity<Pupil>());

_disciplineModelBuilder.Build(modelBu
ilder.Entity<Discipline>());

_subjectModelBuilder.Build(modelBuild
er.Entity<Subject>());
        }
    }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Pupils
{
    public class CreatePupilCommand :
ICommand
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
        public Guid ClassId { get; set; }
        public Guid SchoolId { get; set; }
    }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Pupils
{
    public class DeletePupilCommand :
ICommand
    {
        public Guid Id { get; set; }
    }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Pupils
{
    public class UpdatePupilCommand :
ICommand
    {
        public Guid Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Login { get; set; }
        public Guid? ClassId { get; set; }
    }
}
using System.Threading.Tasks;
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Pupils;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Pupils
```

```
{
    public class
CreatePupilsCommandHandler :
CommandHandlerBase<CreatePupilCom
mand>
    {
        public
CreatePupilsCommandHandler(SchoolS
ystemManagementDbContext
dbContext) : base(dbContext)
        {
        }

        public override async Task
Execute(CreatePupilCommand
command)
        {
            var pupil =
Mapper.Map<Pupil>(command);

            _dbContext.Pupils.Add(pupil);

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Pupils;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Pupils
```

```
{
    public class
DeletePupilCommandHandler :
CommandHandlerBase<DeletePupilCom
mand>
    {
        public
DeletePupilCommandHandler(SchoolSy
stemManagementDbContext dbContext)
: base(dbContext)
        {
        }

        public override async Task
Execute(DeletePupilCommand
command)
        {
            var pupilToDelete = await
_dbContext.Pupils.FirstOrDefaultAsync(
teacher => teacher.Id == command.Id);

            if (pupilToDelete == null)
            {
                throw new
EntityNotFoundException<Teacher,
DeletePupilCommand>(command);
            }


_dbContext.Pupils.Remove(pupilToDele
te);

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
```

```
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Pupils;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Pupils
{
   public class
UpdatePupilCommandHandler :
CommandHandlerBase<UpdatePupilCo
mmand>
   {
      public
UpdatePupilCommandHandler(SchoolSy
stemManagementDbContext dbContext)
: base(dbContext)
      {
      }

      public override async Task
Execute(UpdatePupilCommand
command)
      {
         var pupilToUpdate = await
_dbContext.Pupils.FirstOrDefaultAsync(
teacher => teacher.Id == command.Id);

         if (pupilToUpdate == null)
         {
            throw new
EntityNotFoundException<Teacher,
UpdatePupilCommand>(command);
         }

         pupilToUpdate.Login =
command.Login ??
pupilToUpdate.Login;
         pupilToUpdate.FirstName =
command.FirstName ??
pupilToUpdate.FirstName;
         pupilToUpdate.LastName =
command.LastName ??
pupilToUpdate.LastName;
         pupilToUpdate.ClassId =
command.ClassId ??
pupilToUpdate.ClassId;

         await
_dbContext.SaveChangesAsync();
      }
   }
}
using System.Threading.Tasks;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Context;

namespace
SchoolSystem.Management.Handlers.Co
mmands
{
   public abstract class
CommandHandlerBase<TCommand> :
ICommandHandler<TCommand>
      where TCommand : ICommand
   {
      protected readonly
SchoolSystemManagementDbContext
_dbContext;

      protected
CommandHandlerBase(SchoolSystemM
anagementDbContext dbContext)
      {
         _dbContext = dbContext;
      }

      public abstract Task
Execute(TCommand command);
```

```
    }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.Q
ueries.Subjects
{
    public class SubjectByIdQuery :
IQuery<SubjectByIdQueryResult>
    {
        public Guid Id { get; set; }
    }
}
using System;
using System.Collections.Generic;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.Q
ueries.Subjects
{
    public class SubjectByIdQueryResult :
IQueryResult
    {
        public Guid Id { get; set; }
        public ClassModel Class { get; set;
}
        public TeacherModel Teacher { get;
set; }
        public DisciplineModel Discipline {
get; set; }
        public IEnumerable<LessonModel>
Lessons { get; set; }

        public class ClassModel
        {
            public Guid Id { get; set; }
            public string Name { get; set; }
        }

        public class TeacherModel
        {
            public Guid Id { get; set; }
            public string Name { get; set; }
        }

        public class DisciplineModel
        {
            public Guid Id { get; set; }
            public string Name { get; set; }
        }

        public class LessonModel
        {
            public Guid Id { get; set; }
            public string DayOfTheWeek {
get; set; }
            public short Order { get; set; }
            public short Room { get; set; }
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.Q
ueries;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.Q
ueries.Subjects;

namespace
SchoolSystem.Management.Handlers.Qu
eries.Subjects
{
```

```
    public class
SubjectByIdQueryHandler :
QueryHandlerBase<SubjectByIdQuery,
SubjectByIdQueryResult>
    {
        public
SubjectByIdQueryHandler(SchoolSyste
mManagementDbContext dbContext) :
base(dbContext)
        {
        }

        public override async
Task<SubjectByIdQueryResult>
Execute(SubjectByIdQuery query)
        {
            var requestedSubject =
_dbContext.Subjects
                .Where(subject => subject.Id
== query.Id)
                .Include(subject =>
subject.Discipline)
                .Include(subject =>
subject.Class)
                .Include(subject =>
subject.Teacher)
                .Include(subject =>
subject.Lessons)
                .ThenInclude(lesson =>
lesson.ScheduleCell)
                .ThenInclude(cell =>
cell.DayOfTheWeek)

.Select(Mapper.Map<SubjectByIdQuery
Result>)
                .FirstOrDefault();

            if (requestedSubject == null)
            {
                throw new
EntityNotFoundException<Subject,
SubjectByIdQuery,
SubjectByIdQueryResult>(query);
            }

            return requestedSubject;
        }
    }
}
using System.Threading.Tasks;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Context;

namespace
SchoolSystem.Management.Handlers.Qu
eries
{
    public abstract class
QueryHandlerBase<TQuery,
TQueryResult> :
IQueryHandler<TQuery, TQueryResult>
        where TQuery :
IQuery<TQueryResult>
        where TQueryResult : IQueryResult
    {
        protected readonly
SchoolSystemManagementDbContext
_dbContext;

        protected
QueryHandlerBase(SchoolSystemManag
ementDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        public abstract
Task<TQueryResult> Execute(TQuery
query);
    }
}
using FluentValidation;
```

```
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Pupils;

namespace
SchoolSystem.Management.Validators.C
ommands.Pupils
{
   public class
CreatePupilCommandValidator :
AbstractValidator<CreatePupilCommand
>
   {
      public
CreatePupilCommandValidator(SchoolS
ystemManagementDbContext
dbContext)
      {
         RuleFor(command =>
command.FirstName).NotEmpty();
         RuleFor(command =>
command.LastName).NotEmpty();
         RuleFor(command =>
command.Login).NotEmpty();

         RuleFor(command =>
command.Login).NotEmpty().MustAsyn
c((login, token) =>

dbContext.Pupils.AllAsync(pupil =>
pupil.Login != login, token))
            .WithMessage("Pupil with
such login already exist");

         RuleFor(command =>
command.ClassId).NotEmpty().MustAsy
nc((classId, token) =>

dbContext.Classes.AnyAsync(@class =>
@class.Id == classId))
            .WithMessage("Selected class
doesn't exist");

         RuleFor(command =>
command.SchoolId).NotEmpty().MustA
sync((schoolId, token) =>

dbContext.Schools.AnyAsync(school =>
school.Id == schoolId))
            .WithMessage("Selected
school doesn't exist");
      }
   }
}
using FluentValidation;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Pupils;

namespace
SchoolSystem.Management.Validators.C
ommands.Pupils
{
   public class
UpdatePupilCommandValidator :
AbstractValidator<UpdatePupilComman
d>
   {
      public
UpdatePupilCommandValidator(SchoolS
ystemManagementDbContext
dbContext)
      {
         RuleFor(command =>
command.Id).NotEmpty();
```

```
        RuleFor(command =>
command).MustAsync(async (command,
token) =>
        {
            if (command.Login == null)
            {
                return true;
            }

            var pupilWithSameLogin =
                await
dbContext.Pupils.FirstOrDefaultAsync(p
upil => pupil.Login ==
command.Login);

            return pupilWithSameLogin
== null || pupilWithSameLogin.Id ==
command.Id;
        }).WithMessage("Pupil with such
login already exist");

        RuleFor(command =>
command.ClassId).MustAsync(async
(classId, token) => classId == null

|| (await dbContext.Classes

.AnyAsync(@class =>

@class.Id == classId)))
            .WithMessage("Selected class
doesn't exist");
      }
   }
}
using System;
using Autofac;
using
Autofac.Extensions.DependencyInjectio
n;
using AutoMapper;
using CryptographyService;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using
Microsoft.Extensions.Configuration;
using
Microsoft.Extensions.DependencyInjecti
on;
using SchoolSystem.Application;
using SchoolSystem.Exceptions;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Profiles;
using
Swashbuckle.AspNetCore.Swagger;

namespace SchoolSystem.Management
{
   public class Startup
   {
      public Startup(IConfiguration
configuration)
      {
         Configuration = configuration;
      }

      public IConfiguration Configuration
{ get; }

      public IServiceProvider
ConfigureServices(IServiceCollection
services)
      {

services.AddMvc().SetCompatibilityVer
sion(CompatibilityVersion.Version_2_2)
;
```

```
services.AddSwaggerGen(generationOpt
ions =>
        {

generationOptions.SwaggerDoc("v1",
new Info {Title = "Administration"});
        });

        var connectionString =
Configuration.GetValue<string>("Conne
ctionString");


services.AddDbContext<SchoolSystem
ManagementDbContext>(optionsBuilder
=>

optionsBuilder.UseSqlServer(connection
String));

        services.AddCors(corsOptions =>

corsOptions.AddPolicy("default", builder
=> builder
                .AllowAnyOrigin()
                .AllowAnyHeader()

.AllowAnyMethod().Build()));


Mapper.Initialize(mapperConfiguration
=>
        {

mapperConfiguration.AddProfile<ClassP
rofile>();

mapperConfiguration.AddProfile<Teach
erProfile>();

mapperConfiguration.AddProfile<PupilP
rofile>();

mapperConfiguration.AddProfile<Discip
lineProfile>();

mapperConfiguration.AddProfile<Subjec
tProfile>();

mapperConfiguration.AddProfile<Lesso
nProfile>();
        });

        var containerBuilder = new
ContainerBuilder();


containerBuilder.Populate(services);

containerBuilder.RegisterModule<Crypt
ographyModule>();

containerBuilder.RegisterModule<Appli
cationModule>();

containerBuilder.RegisterModule<Mana
gementModule>();

        return new
AutofacServiceProvider(containerBuilde
r.Build());
    }

    public void
Configure(IApplicationBuilder app,
IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {

app.UseDeveloperExceptionPage();
```

```
            }


app.UseMiddleware<ExceptionHandling
Middleware>();

            app.UseMvc();

            app.UseSwagger();


app.UseSwaggerUI(swaggerOptions =>
            {

swaggerOptions.SwaggerEndpoint("/swa
gger/v1/swagger.json", "My API V1");
                swaggerOptions.RoutePrefix =
string.Empty;
            });
        }
    }
}
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Contracts.C
ommands.Pupils;
using
SchoolSystem.Management.Contracts.Q
ueries.Pupils;

namespace
SchoolSystem.Management.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PupilsController :
ControllerBase
    {
        private readonly ICommandBus
_commandBus;
        private readonly IQueryBus
_queryBus;

        public
PupilsController(ICommandBus
commandBus, IQueryBus queryBus)
        {
            _commandBus = commandBus;
            _queryBus = queryBus;
        }

        [HttpGet]
        public async
Task<PupilsQueryResult> Get()
        {
            return await
_queryBus.Execute<PupilsQuery,
PupilsQueryResult>(new
PupilsQuery());
        }

        [HttpPost]
        public async Task Post([FromBody]
CreatePupilCommand command)
        {
            await
_commandBus.Execute(command);
        }

        [HttpPut]
        public async Task
Put(UpdatePupilCommand command)
        {
            await
_commandBus.Execute(command);
        }

        [HttpDelete("{id}")]
        public async Task Delete(Guid id)
        {
```

```
            await
_commandBus.Execute(new
DeletePupilCommand
            {
                Id = id
            });
        }
    }
}
using
Microsoft.EntityFrameworkCore.Metada
ta.Builders;
using SchoolSystem.Entities;

namespace
SchoolSystem.Management.Context.Mo
delBuilders.Abstractions
{
    public abstract class
ModelBuilderBase<TEntity> :
IModelBuilder<TEntity>
        where TEntity : EntityBase
    {
        public void
Build(EntityTypeBuilder<TEntity>
entityTypeBuilder)
        {
            entityTypeBuilder
                .HasKey(user => user.Id);

            entityTypeBuilder
                .Property(entity =>
entity.Id).ValueGeneratedOnAdd();

            BuildModel(entityTypeBuilder);
        }

        protected abstract void
BuildModel(EntityTypeBuilder<TEntity
> entityTypeBuilder);
    }
}
```

```
using
Microsoft.EntityFrameworkCore.Metada
ta.Builders;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context.Mo
delBuilders.Abstractions;

namespace
SchoolSystem.Management.Context.Mo
delBuilders.Implementation
{
    public class LessonModelBuilder :
ModelBuilderBase<Lesson>
    {
        protected override void
BuildModel(EntityTypeBuilder<Lesson>
entityTypeBuilder)
        {

entityTypeBuilder.HasIndex(lesson =>
new {lesson.SchoolId,
lesson.ScheduleCellId}).IsUnique();

entityTypeBuilder.HasIndex(lesson =>
lesson.SubjectId);

entityTypeBuilder.HasIndex(lesson =>
lesson.ScheduleCellId);
        }
    }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Subjects
{
    public class CreateSubjectCommand :
ICommand
    {
```

```
        public Guid ClassId { get; set; }
        public Guid TeacherId { get; set; }
        public Guid DisciplineId { get; set;
}
        public Guid SchoolId { get; set; }
    }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Subjects
{
    public class DeleteSubjectCommand :
ICommand
    {
        public Guid Id { get; set; }
    }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Subjects
{
    public class UpdateSubjectCommand :
ICommand
    {
        public Guid Id { get; set; }
        public Guid? ClassId { get; set; }
        public Guid? TeacherId { get; set; }
        public Guid? DisciplineId { get; set;
}
    }
}
using System.Threading.Tasks;
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Subjects;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Subjects
{
    public class
CreateSubjectCommandHandler :
CommandHandlerBase<CreateSubjectC
ommand>
    {
        public
CreateSubjectCommandHandler(School
SystemManagementDbContext
dbContext) : base(dbContext)
        {
        }

        public override async Task
Execute(CreateSubjectCommand
command)
        {
            var subject =
Mapper.Map<Subject>(command);


_dbContext.Subjects.Add(subject);

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
```

```
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Subjects;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Subjects
{
   public class
DeleteSubjectCommandHandler :
CommandHandlerBase<DeleteSubjectC
ommand>
   {
      public
DeleteSubjectCommandHandler(School
SystemManagementDbContext
dbContext) : base(dbContext)
      {
      }

      public override async Task
Execute(DeleteSubjectCommand
command)
      {
         var subjectToDelete = await
_dbContext.Subjects.FirstOrDefaultAsyn
c(subject => subject.Id == command.Id);

         if (subjectToDelete == null)
         {
            throw new
EntityNotFoundException<Subject,
DeleteSubjectCommand>(command);
         }


_dbContext.Subjects.Remove(subjectTo
Delete);
         await
_dbContext.SaveChangesAsync();
      }
   }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Subjects;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Subjects
{
   public class
UpdateSubjectCommandHandler :
CommandHandlerBase<UpdateSubjectC
ommand>
   {
      public
UpdateSubjectCommandHandler(School
SystemManagementDbContext
dbContext) : base(dbContext)
      {
      }

      public override async Task
Execute(UpdateSubjectCommand
command)
      {
         var subjectToUpdate = await
_dbContext.Subjects.FirstOrDefaultAsyn
c(subject => subject.Id == command.Id);

         if (subjectToUpdate == null)
```

```
            {
                throw new
EntityNotFoundException<Subject,
UpdateSubjectCommand>(command);
            }

            subjectToUpdate.ClassId =
command.ClassId ??
subjectToUpdate.ClassId;
            subjectToUpdate.TeacherId =
command.TeacherId ??
subjectToUpdate.TeacherId;
            subjectToUpdate.DisciplineId =
command.DisciplineId ??
subjectToUpdate.DisciplineId;

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.Q
ueries.Subjects
{
    public class SubjectsQuery :
IQuery<SubjectsQueryResult>
    {
    }
}
using System;
using System.Collections.Generic;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.Q
ueries.Subjects
{
    public class SubjectsQueryResult:
IQueryResult
    {
        public
IEnumerable<SubjectsQueryResultMode
l> Subjects { get; set; }
    }

    public class
SubjectsQueryResultModel
    {
        public Guid Id { get; set; }
        public string Discipline { get; set; }
        public string Class { get; set; }
        public string Teacher { get; set; }
    }
}
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.Q
ueries.Subjects;

namespace
SchoolSystem.Management.Handlers.Qu
eries.Subjects
{
    public class SubjectsQueryHandler :
QueryHandlerBase<SubjectsQuery,
SubjectsQueryResult>
    {
        public
SubjectsQueryHandler(SchoolSystemMa
nagementDbContext dbContext) :
base(dbContext)
        {
        }
```

```
        public override async
Task<SubjectsQueryResult>
Execute(SubjectsQuery query)
        {
            var subjects = await
_dbContext.Subjects
                .Include(subject =>
subject.Class)
                .Include(subject =>
subject.Teacher)
                .Include(subject =>
subject.Discipline)
                .Select(subject =>
Mapper.Map<SubjectsQueryResultMode
l>(subject))
                .ToListAsync();

            return new SubjectsQueryResult
            {
                Subjects = subjects
            };
        }
    }
}
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Contracts.C
ommands.Subjects;
using
SchoolSystem.Management.Contracts.Q
ueries.Subjects;
using
SchoolSystem.Management.Profiles.Con
verters.Subjects;

namespace
SchoolSystem.Management.Profiles
{
    public class SubjectProfile : Profile
    {
        public SubjectProfile()
        {

CreateMap<CreateSubjectCommand,
Subject>();
            CreateMap<Subject,
SubjectsQueryResultModel>().ConvertU
sing<SubjectToSubjectsQueryResultMo
delConverter>();
            CreateMap<Subject,
SubjectByIdQueryResult>().ConvertUsi
ng<SubjectToSubjectByIdQueryResultC
onverter>();
        }
    }
}
using System.Collections.Generic;
using System.Linq;
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Contracts.Q
ueries.Subjects;

namespace
SchoolSystem.Management.Profiles.Con
verters.Subjects
{
    public class
SubjectToSubjectByIdQueryResultConv
erter : ITypeConverter<Subject,
SubjectByIdQueryResult>
    {
        public SubjectByIdQueryResult
Convert(Subject source,
SubjectByIdQueryResult destination,
            ResolutionContext context)
        {
            var lessons =
source.Lessons.Select(Mapper.Map<Sub
jectByIdQueryResult.LessonModel>);
```

```
            return new
SubjectByIdQueryResult
            {
                Id = source.Id,
                Class = new
SubjectByIdQueryResult.ClassModel
                {
                    Id = source.Class.Id,
                    Name = source.Class.Name
                },
                Discipline = new
SubjectByIdQueryResult.DisciplineMod
el
                {
                    Id = source.Discipline.Id,
                    Name =
source.Discipline.Name
                },
                Teacher = new
SubjectByIdQueryResult.TeacherModel
                {
                    Id = source.Teacher.Id,
                    Name =
$"{source.Teacher.LastName}
{source.Teacher.FirstName}"
                },
                Lessons = lessons
            };
        }
    }
}
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Contracts.Q
ueries.Subjects;

namespace
SchoolSystem.Management.Profiles.Con
verters.Subjects
{
    public class
SubjectToSubjectsQueryResultModelCo
nverter : ITypeConverter<Subject,
SubjectsQueryResultModel>
    {
        public SubjectsQueryResultModel
Convert(Subject source,
SubjectsQueryResultModel destination,
            ResolutionContext context)
        {
            return new
SubjectsQueryResultModel
            {
                Id = source.Id,
                Class = source.Class.Name,
                Discipline =
source.Discipline.Name,
                Teacher =
$"{source.Teacher.LastName}
{source.Teacher.FirstName}"
            };
        }
    }
}
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Contracts.C
ommands.Lessons;
using
SchoolSystem.Management.Contracts.Q
ueries.Subjects;
using
SchoolSystem.Management.Profiles.Con
verters.Lessons;

namespace
SchoolSystem.Management.Profiles
{
    public class LessonProfile : Profile
    {
        public LessonProfile()
```

```
        {

CreateMap<CreateLessonCommand,
Lesson>();
            CreateMap<Lesson,
SubjectByIdQueryResult.LessonModel>
().ConvertUsing<LessonToLessonModel
>();
        }
    }
}
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Contracts.Q
ueries.Subjects;

namespace
SchoolSystem.Management.Profiles.Con
verters.Lessons
{
    public class LessonToLessonModel :
ITypeConverter<Lesson,

SubjectByIdQueryResult.LessonModel>
    {
        public
SubjectByIdQueryResult.LessonModel
Convert(Lesson source,

SubjectByIdQueryResult.LessonModel
destination, ResolutionContext context)
        {
            var lesson = source;
            var lessonScheduleCell =
lesson.ScheduleCell;
            var dayOfTheWeek =
lessonScheduleCell?.DayOfTheWeek;

            return new
SubjectByIdQueryResult.LessonModel
            {
                Id = lesson.Id,
                DayOfTheWeek =
lesson.ScheduleCell.DayOfTheWeek.Na
me,
                Order =
lesson.ScheduleCell.Order,
                Room = lesson.Room
            };
        }
    }
}
using Autofac;
using FluentValidation;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Context.Mo
delBuilders;

namespace SchoolSystem.Management
{
    public class ManagementModule :
Module
    {
        protected override void
Load(ContainerBuilder builder)
        {

builder.RegisterAssemblyTypes(ThisAss
embly)

.AsClosedTypesOf(typeof(IQueryHandle
r<,>))
            .AsImplementedInterfaces();

builder.RegisterAssemblyTypes(ThisAss
embly)

.AsClosedTypesOf(typeof(ICommandHa
ndler<>))
            .AsImplementedInterfaces();
```

```
builder.RegisterAssemblyTypes(ThisAss
embly)

.AsClosedTypesOf(typeof(IValidator<>)
)
            .AsImplementedInterfaces();


builder.RegisterModule<ModelBuilderM
odule>();
      }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Classes
{
   public class CreateClassCommand :
ICommand
   {
      public string Name { get; set; }
      public Guid SchoolId { get; set; }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Classes
{
   public class DeleteClassCommand :
ICommand
   {
      public Guid Id { get; set; }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Classes
{
   public class UpdateClassCommand :
ICommand
   {
      public Guid Id { get; set; }
      public string Name { get; set; }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Disciplines
{
   public class
CreateDisciplineCommand : ICommand
   {
      public string Name { get; set; }
      public Guid SchoolId { get; set; }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Disciplines
{
   public class
DeleteDisciplineCommand : ICommand
   {
      public Guid Id { get; set; }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;
```

```
namespace
SchoolSystem.Management.Contracts.C
ommands.Disciplines
{
   public class
UpdateDisciplineCommand : ICommand
   {
      public Guid Id { get; set; }
      public string Name { get; set; }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Teachers
{
   public class CreateTeacherCommand :
ICommand
   {
      public string FirstName { get; set; }
      public string LastName { get; set; }
      public string Login { get; set; }
      public string Password { get; set; }
      public Guid SchoolId { get; set; }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Teachers
{
   public class DeleteTeacherCommand :
ICommand
   {
      public Guid Id { get; set; }
   }
}
using System;
using SchoolSystem.Abstractions.CQRS;

namespace
SchoolSystem.Management.Contracts.C
ommands.Teachers
{
   public class UpdateTeacherCommand
: ICommand
   {
      public Guid Id { get; set; }
      public string FirstName { get; set; }
      public string LastName { get; set; }
      public string Login { get; set; }
   }
}
using System.Threading.Tasks;
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Classes;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Classes
{
   public class
CreateClassCommandHandler :
CommandHandlerBase<CreateClassCom
mand>
   {
      public
CreateClassCommandHandler(SchoolSy
stemManagementDbContext dbContext)
: base(dbContext)
      {
      }
```

```
        public override async Task
Execute(CreateClassCommand
command)
        {
            var @class =
Mapper.Map<Class>(command);


_dbContext.Classes.Add(@class);

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Classes;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Classes
{
    public class
DeleteClassCommandHandler :
CommandHandlerBase<DeleteClassCom
mand>
    {
        public
DeleteClassCommandHandler(SchoolSy
stemManagementDbContext dbContext)
: base(dbContext)
        {
        }
```

```
        public override async Task
Execute(DeleteClassCommand
command)
        {
            var classToDelete = await
_dbContext.Classes.FirstOrDefaultAsync
(@class => @class.Id == command.Id);

            if (classToDelete == null)
            {
                throw new
EntityNotFoundException<Class,
DeleteClassCommand>(command);
            }


_dbContext.Remove(classToDelete);

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Classes;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Classes
{
    public class
UpdateClassCommandHandler :
```

```
CommandHandlerBase<UpdateClassCo
mmand>
   {
      public
UpdateClassCommandHandler(SchoolSy
stemManagementDbContext dbContext)
: base(dbContext)
      {
      }

      public override async Task
Execute(UpdateClassCommand
command)
      {
         var classToUpdate = await
_dbContext.Classes.FirstOrDefaultAsync
(@class => @class.Id == command.Id);

         if (classToUpdate == null)
         {
            throw new
EntityNotFoundException<Class,
UpdateClassCommand>(command);
         }

         classToUpdate.Name =
command.Name ??
classToUpdate.Name;

         await
_dbContext.SaveChangesAsync();
      }
   }
}
using System.Threading.Tasks;
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Disciplines;
```

```
namespace
SchoolSystem.Management.Handlers.Co
mmands.Disciplines
{
   public class
CreateDisciplineCommandHandler :
CommandHandlerBase<CreateDiscipline
Command>
   {
      public
CreateDisciplineCommandHandler(Scho
olSystemManagementDbContext
dbContext) : base(dbContext)
      {
      }

      public override async Task
Execute(CreateDisciplineCommand
command)
      {
         var discipline =
Mapper.Map<Discipline>(command);


_dbContext.Disciplines.Add(discipline);

         await
_dbContext.SaveChangesAsync();
      }
   }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
```

```
using
SchoolSystem.Management.Contracts.C
ommands.Disciplines;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Disciplines
{
   public class
DeleteDisciplineCommandHandler :
CommandHandlerBase<DeleteDiscipline
Command>
   {
      public
DeleteDisciplineCommandHandler(Scho
olSystemManagementDbContext
dbContext) : base(dbContext)
      {
      }

      public override async Task
Execute(DeleteDisciplineCommand
command)
      {
         var disciplineToDelete =
            await
_dbContext.Disciplines.FirstOrDefaultA
sync(discipline => discipline.Id ==
command.Id);

         if (disciplineToDelete == null)
         {
            throw new
EntityNotFoundException<Discipline,
DeleteDisciplineCommand>(command);
         }


_dbContext.Remove(disciplineToDelete)
;
         await
_dbContext.SaveChangesAsync();
      }
   }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Disciplines;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Disciplines
{
   public class
UpdateDisciplineCommandHandler :
CommandHandlerBase<UpdateDisciplin
eCommand>
   {
      public
UpdateDisciplineCommandHandler(Sch
oolSystemManagementDbContext
dbContext) : base(dbContext)
      {
      }

      public override async Task
Execute(UpdateDisciplineCommand
command)
      {
         var disciplineToUpdate =
            await
_dbContext.Disciplines.FirstOrDefaultA
sync(discipline => discipline.Id ==
command.Id);
```

```
            if (disciplineToUpdate == null)
            {
                throw new
EntityNotFoundException<Discipline,
UpdateDisciplineCommand>(command);
            }

            disciplineToUpdate.Name =
command.Name ??
disciplineToUpdate.Name;

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Teachers;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Teachers
{
    public class
CreateTeacherCommandHandler :
CommandHandlerBase<CreateTeacherC
ommand>
    {
        public
CreateTeacherCommandHandler(School
SystemManagementDbContext
dbContext) : base(dbContext)
        {
        }

        public override async Task
Execute(CreateTeacherCommand
command)
        {
            var teacher =
Mapper.Map<Teacher>(command);


_dbContext.Teachers.Add(teacher);

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Teachers;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Teachers
{
    public class
DeleteTeacherCommandHandler :
CommandHandlerBase<DeleteTeacherC
ommand>
    {
        public
DeleteTeacherCommandHandler(School
SystemManagementDbContext
dbContext) : base(dbContext)
        {
        }
```

```
        public override async Task
Execute(DeleteTeacherCommand
command)
        {
            var teacherToDelete = await
_dbContext.Teachers.FirstOrDefaultAsy
nc(teacher => teacher.Id ==
command.Id);

            if (teacherToDelete == null)
            {
                throw new
EntityNotFoundException<Teacher,
DeleteTeacherCommand>(command);
            }


_dbContext.Teachers.Remove(teacherTo
Delete);

            await
_dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Application.Exceptions.C
ommands;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Teachers;

namespace
SchoolSystem.Management.Handlers.Co
mmands.Teachers
{
    public class
UpdateTeacherCommandHandler :
CommandHandlerBase<UpdateTeacher
Command>
    {
        public
UpdateTeacherCommandHandler(School
SystemManagementDbContext
dbContext) : base(dbContext)
        {
        }

        public override async Task
Execute(UpdateTeacherCommand
command)
        {
            var teacherToUpdate = await
_dbContext.Teachers.FirstOrDefaultAsy
nc(teacher => teacher.Id ==
command.Id);

            if (teacherToUpdate == null)
            {
                throw new
EntityNotFoundException<Teacher,
UpdateTeacherCommand>(command);
            }

            teacherToUpdate.Login =
command.Login ??
teacherToUpdate.Login;
            teacherToUpdate.FirstName =
command.FirstName ??
teacherToUpdate.FirstName;
            teacherToUpdate.LastName =
command.LastName ??
teacherToUpdate.LastName;

            await
_dbContext.SaveChangesAsync();
        }
    }
```

```
}
using FluentValidation;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Classes;

namespace
SchoolSystem.Management.Validators.C
ommands.Classes
{
   public class
CreateClassCommandValidator :
AbstractValidator<CreateClassComman
d>
   {
      public
CreateClassCommandValidator(SchoolS
ystemManagementDbContext
dbContext)
      {
         RuleFor(command =>
command.Name).NotEmpty().MustAsyn
c((name, token) =>

dbContext.Classes.AllAsync(@class =>
@class.Name != name, token))
            .WithMessage("Class with
such name already exist");

         RuleFor(command =>
command.SchoolId).NotEmpty().MustA
sync((schoolId, token) =>

dbContext.Schools.AnyAsync(school =>
school.Id == schoolId))
            .WithMessage("Selected
school doesn't exist");
      }
   }
```

```
}
using FluentValidation;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Classes;

namespace
SchoolSystem.Management.Validators.C
ommands.Classes
{
   public class
UpdateClassCommandValidator :
AbstractValidator<UpdateClassComman
d>
   {
      public
UpdateClassCommandValidator(School
SystemManagementDbContext
dbContext)
      {
         RuleFor(command =>
command.Id).NotEmpty();
         RuleFor(command =>
command).MustAsync(async (command,
token) =>
         {
            var classWithSameName =
               await
dbContext.Classes.FirstOrDefaultAsync(
@class => @class.Name ==
command.Name);

            return classWithSameName ==
null || classWithSameName.Id ==
command.Id;
         }).WithMessage("Class with such
name already exist");
      }
   }
```

```
}
using FluentValidation;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Disciplines;

namespace
SchoolSystem.Management.Validators.C
ommands.Disciplines
{
   public class
CreateDisciplineCommandValidator :
AbstractValidator<CreateDisciplineCom
mand>
   {
      public
CreateDisciplineCommandValidator(Sch
oolSystemManagementDbContext
dbContext)
      {
         RuleFor(command =>
command).MustAsync(async (command,
token) =>
         {
            var disciplineWithSameName
=
               await
dbContext.Disciplines.FirstOrDefaultAs
ync(discipline => discipline.Name ==
command.Name);

            return
disciplineWithSameName == null ||
disciplineWithSameName.SchoolId !=
command.SchoolId;
         }).WithMessage("Disciplien with
such name already exist in this school");
         RuleFor(command =>
command.SchoolId).NotEmpty().MustA
sync((schoolId, token) =>

dbContext.Schools.AnyAsync(school =>
school.Id == schoolId))
            .WithMessage("Selected
school doesn't exist");
      }
   }
}
using FluentValidation;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Disciplines;

namespace
SchoolSystem.Management.Validators.C
ommands.Disciplines
{
   public class
UpdateDisciplineCommandValidator :
AbstractValidator<UpdateDisciplineCo
mmand>
   {
      public
UpdateDisciplineCommandValidator(Sc
hoolSystemManagementDbContext
dbContext)
      {
         RuleFor(command =>
command.Id).NotEmpty();

         RuleFor(command =>
command).MustAsync(async (command,
token) =>
         {
            if (command.Name == null)
            {
```

```
                return true;
            }

            var disciplineWithSameName
=
                await
dbContext.Disciplines.FirstOrDefaultAs
ync(discipline => discipline.Name ==
command.Name);

            return
disciplineWithSameName == null ||
disciplineWithSameName.Id ==
command.Id;
        }).WithMessage("Disciplien with
such name already exist in this school");
    }
  }
}
using FluentValidation;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Teachers;

namespace
SchoolSystem.Management.Validators.C
ommands.Teachers
{
  public class
CreateTeacherCommandValidator :
AbstractValidator<CreateTeacherComm
and>
  {
    public
CreateTeacherCommandValidator(Schoo
lSystemManagementDbContext
dbContext)
    {
        RuleFor(command =>
command.FirstName).NotEmpty();
        RuleFor(command =>
command.LastName).NotEmpty();
        RuleFor(command =>
command.Login).NotEmpty();

        RuleFor(command =>
command.Login).NotEmpty().MustAsyn
c((login, token) =>

dbContext.Teachers.AllAsync(teacher
=> teacher.Login != login, token))
            .WithMessage("Teacher with
such login already exist");

        RuleFor(command =>
command.SchoolId).NotEmpty().MustA
sync((schoolId, token) =>

dbContext.Schools.AnyAsync(school =>
school.Id == schoolId))
            .WithMessage("Selected
school doesn't exist");
    }
  }
}
using FluentValidation;
using Microsoft.EntityFrameworkCore;
using
SchoolSystem.Management.Context;
using
SchoolSystem.Management.Contracts.C
ommands.Teachers;

namespace
SchoolSystem.Management.Validators.C
ommands.Teachers
{
  public class
UpdateTeacherCommandValidator :
```

```
AbstractValidator<UpdateTeacherComm
and>
   {
      public
UpdateTeacherCommandValidator(Scho
olSystemManagementDbContext
dbContext)
      {
         RuleFor(command =>
command.Id).NotEmpty();
         RuleFor(command =>
command).MustAsync(async (command,
token) =>
         {
            if (command.Login == null)
            {
               return true;
            }

            var classWithSameName =
               await
dbContext.Teachers.FirstOrDefaultAsyn
c(teacher => teacher.Login ==
command.Login);

            return classWithSameName ==
null || classWithSameName.Id ==
command.Id;
         }).WithMessage("Teacher with
such login already exist");
      }
   }
}
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Contracts.C
ommands.Classes;
using
SchoolSystem.Management.Contracts.Q
ueries.Classes;

namespace
SchoolSystem.Management.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class ClassesController :
ControllerBase
   {
      private readonly ICommandBus
_commandBus;
      private readonly IQueryBus
_queryBus;

      public
ClassesController(ICommandBus
commandBus, IQueryBus queryBus)
      {
         _commandBus = commandBus;
         _queryBus = queryBus;
      }

      [HttpGet]
      public async
Task<ClassesQueryResult> Get()
      {
         return await
_queryBus.Execute<ClassesQuery,
ClassesQueryResult>(new
ClassesQuery());
      }

      [HttpPost]
      public async Task Post([FromBody]
CreateClassCommand command)
      {
         await
_commandBus.Execute(command);
      }
```

```
        [HttpPut]
        public async Task
Put(UpdateClassCommand command)
        {
            await
_commandBus.Execute(command);
        }

        [HttpDelete("{id}")]
        public async Task Delete(Guid id)
        {
            await
_commandBus.Execute(new
DeleteClassCommand
            {
                Id = id
            });
        }
    }
}
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Contracts.C
ommands.Disciplines;
using
SchoolSystem.Management.Contracts.Q
ueries.Disciplines;

namespace
SchoolSystem.Management.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class DisciplinesController :
ControllerBase
    {
        private readonly ICommandBus
_commandBus;
        private readonly IQueryBus
_queryBus;

        public
DisciplinesController(ICommandBus
commandBus, IQueryBus queryBus)
        {
            _commandBus = commandBus;
            _queryBus = queryBus;
        }

        [HttpGet]
        public async
Task<DisciplinesQueryResult> Get()
        {
            return await
_queryBus.Execute<DisciplinesQuery,
DisciplinesQueryResult>(new
DisciplinesQuery());
        }

        [HttpPost]
        public async Task Post([FromBody]
CreateDisciplineCommand command)
        {
            await
_commandBus.Execute(command);
        }

        [HttpPut]
        public async Task
Put(UpdateDisciplineCommand
command)
        {
            await
_commandBus.Execute(command);
        }

        [HttpDelete("{id}")]
        public async Task Delete(Guid id)
        {
```

```
            await
_commandBus.Execute(new
DeleteDisciplineCommand
            {
                Id = id
            });
        }
    }
}
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Contracts.C
ommands.Lessons;

namespace
SchoolSystem.Management.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class LessonsController :
ControllerBase
    {
        private readonly ICommandBus
_commandBus;
        private readonly IQueryBus
_queryBus;

        public
LessonsController(ICommandBus
commandBus, IQueryBus queryBus)
        {
            _commandBus = commandBus;
            _queryBus = queryBus;
        }

        //[HttpGet]
        //public async
Task<ClassesQueryResult> Get()
        //{
        //    return await
_queryBus.Execute<ClassesQuery,
ClassesQueryResult>(new
ClassesQuery());
        //}

        [HttpPost]
        public async Task Post([FromBody]
CreateLessonCommand command)
        {
            await
_commandBus.Execute(command);
        }

        [HttpPut]
        public async Task
Put(UpdateLessonCommand command)
        {
            await
_commandBus.Execute(command);
        }

        [HttpDelete("{id}")]
        public async Task Delete(Guid id)
        {
            await
_commandBus.Execute(new
DeleteLessonCommand
            {
                Id = id
            });
        }
    }
}
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Contracts.C
ommands.Subjects;
```

```
using SchoolSystem.Management.Contracts.Queries.Subjects;

namespace SchoolSystem.Management.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class SubjectsController : ControllerBase
    {
        private readonly ICommandBus _commandBus;
        private readonly IQueryBus _queryBus;

        public SubjectsController(ICommandBus commandBus, IQueryBus queryBus)
        {
            _commandBus = commandBus;
            _queryBus = queryBus;
        }

        [HttpGet]
        public async Task<SubjectsQueryResult> GetAllSubjects()
        {
            return await _queryBus.Execute<SubjectsQuery, SubjectsQueryResult>(new SubjectsQuery());
        }

        [HttpGet("{id}")]
        public async Task<SubjectByIdQueryResult> GetSubjectById(Guid id)
        {
            return await _queryBus.Execute<SubjectByIdQuery, SubjectByIdQueryResult>(new SubjectByIdQuery
            {
                Id = id
            });
        }

        [HttpPost]
        public async Task CreateSubject(CreateSubjectCommand command)
        {
            await _commandBus.Execute(command);
        }

        [HttpPut]
        public async Task UpdateSubject(UpdateSubjectCommand command)
        {
            await _commandBus.Execute(command);
        }

        [HttpDelete("{id}")]
        public async Task DeleteSubject(Guid id)
        {
            var command = new DeleteSubjectCommand
            {
                Id = id
            };

            await _commandBus.Execute(command);
        }
    }
}
```

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Management.Contracts.C
ommands.Teachers;
using
SchoolSystem.Management.Contracts.Q
ueries.Teachers;

namespace
SchoolSystem.Management.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class TeachersController :
ControllerBase
   {
      private readonly ICommandBus
_commandBus;
      private readonly IQueryBus
_queryBus;

      public
TeachersController(ICommandBus
commandBus, IQueryBus queryBus)
      {
         _commandBus = commandBus;
         _queryBus = queryBus;
      }

      [HttpGet]
      public async
Task<TeachersQueryResult> Get()
      {
         return await
_queryBus.Execute<TeachersQuery,
TeachersQueryResult>(new
TeachersQuery());
      }

      [HttpPost]
      public async Task Post([FromBody]
CreateTeacherCommand command)
      {
         await
_commandBus.Execute(command);
      }

      [HttpPut]
      public async Task
Put(UpdateTeacherCommand command)
      {
         await
_commandBus.Execute(command);
      }

      [HttpDelete("{id}")]
      public async Task Delete(Guid id)
      {
         await
_commandBus.Execute(new
DeleteTeacherCommand
         {
            Id = id
         });
      }
   }
}
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Contracts.C
ommands.Disciplines;
using
SchoolSystem.Management.Contracts.Q
ueries.Disciplines;

namespace
SchoolSystem.Management.Profiles
{
   public class DisciplineProfile : Profile
   {
```

```
        public DisciplineProfile()
        {

CreateMap<CreateDisciplineCommand,
Discipline>();
            CreateMap<Discipline,
DisciplinesQueryResultModel>();
        }
    }
}
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Contracts.C
ommands.Classes;
using
SchoolSystem.Management.Contracts.Q
ueries.Classes;

namespace
SchoolSystem.Management.Profiles
{
    public class ClassProfile : Profile
    {
        public ClassProfile()
        {

CreateMap<CreateClassCommand,
Class>();
            CreateMap<Class,
ClassesQueryResultModel>();
        }
    }
}
using AutoMapper;
using SchoolSystem.Entities;
using
SchoolSystem.Management.Contracts.C
ommands.Teachers;
using
SchoolSystem.Management.Contracts.Q
ueries.Teachers;

namespace
SchoolSystem.Management.Profiles
{
    public class TeacherProfile : Profile
    {
        public TeacherProfile()
        {

CreateMap<CreateTeacherCommand,
Teacher>();
            CreateMap<Teacher,
TeachersQueryResultModel>();
        }
    }
}
using System;

namespace SchoolSystem.Entities
{
    public abstract class EntityBase
    {
        public Guid Id { get; set; }
    }
}
using System;

namespace SchoolSystem.Entities
{
    public abstract class
SchoolScopedEntityBase : EntityBase
    {
        public Guid SchoolId { get; set; }
        public School School { get; set; }
    }
}
using System;

namespace SchoolSystem.Entities
{
    public abstract class User : EntityBase
    {
```

```
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
        public School School { get; set; }
        public Guid SchoolId { get; set; }

        public string RefreshToken { get;
set; }
    }
}

using System;

namespace SchoolSystem.Entities
{
    public class Teacher : User
    {
    }
}
using System;

namespace SchoolSystem.Entities
{
    public class Pupil : User
    {
        public Class Class { get; set; }
        public Guid ClassId { get; set; }
    }
}
namespace SchoolSystem.Entities
{
    public class Discipline :
SchoolScopedEntityBase
    {
        public string Name { get; set; }
    }
}
using System;
using System.Collections.Generic;

namespace SchoolSystem.Entities
{
    public class Subject :
SchoolScopedEntityBase
    {
        public Guid ClassId { get; set; }
        public Class Class { get; set; }
        public Guid TeacherId { get; set; }
        public Teacher Teacher { get; set; }
        public Guid DisciplineId { get; set;
}
        public Discipline Discipline { get;
set; }
        public IEnumerable<Lesson>
Lessons { get; set; }
    }
}
using System;

namespace SchoolSystem.Entities
{
    public class Lesson :
SchoolScopedEntityBase
    {
        public Guid SubjectId { get; set; }
        public Subject Subject { get; set; }
        public Guid ScheduleCellId { get;
set; }
        public ScheduleCell ScheduleCell {
get; set; }
        public short Room { get; set; }
    }
}
using System;

namespace SchoolSystem.Entities
{
    public class ScheduleCell : EntityBase
    {
        public Guid DayOfTheWeekId {
get; set; }
        public DayOfTheWeek
DayOfTheWeek { get; set; }
```

```
        public short Order { get; set; }
    }
}
using System;

namespace SchoolSystem.Entities
{
    public class School : EntityBase
    {
        public string Name { get; set; }
        public Guid CityId { get; set; }
        public City City { get; set; }

        public string Login { get; set; }
        public string Password { get; set; }
        public string RefreshToken { get;
set; }
    }
}
using System.Collections;
using System.Collections.Generic;

namespace SchoolSystem.Entities
{
    public class City : EntityBase
    {
        public string Name { get; set; }
    }
}
using System.Collections.Generic;

namespace SchoolSystem.Entities
{
    public class Class :
SchoolScopedEntityBase
    {
        public string Name { get; set; }
        public IEnumerable<Pupil> Pupils
{ get; set; }
    }
}
namespace SchoolSystem.Entities
{
    public class DayOfTheWeek :
EntityBase
    {
        public string Name { get; set; }
    }
}
using System;
using System.Threading.Tasks;
using Autofac;
using FluentValidation;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Application.Exceptions.C
ommands;
using
SchoolSystem.Application.Exceptions.V
alidation;
using ValidationException =
SchoolSystem.Application.Exceptions.V
alidation.ValidationException;

namespace
SchoolSystem.Application.Buses
{
    internal class CommandBus :
ICommandBus
    {
        private readonly ILifetimeScope
_lifetimeScope;

        public
CommandBus(ILifetimeScope
lifetimeScope)
        {
            _lifetimeScope = lifetimeScope;
        }

        public async Task
Execute<TCommand>(TCommand
command) where TCommand :
ICommand
```

```
        {
            var commandHandler =
ResolveHandler<TCommand>(command
);

            await Validate(command);

            await
commandHandler.Execute(command);
        }

        private
ICommandHandler<TCommand>
ResolveHandler<TCommand>(TComma
nd command)
            where TCommand : ICommand
        {
            var commandHandler =
_lifetimeScope.ResolveOptional<ICom
mandHandler<TCommand>>();

            if (commandHandler == null)
            {
                throw new
CommandHandlerNotFoundException<
TCommand>(command);
            }

            return commandHandler;
        }

        private async Task
Validate<TCommand>(TCommand
command)
            where TCommand : ICommand
        {
            var validator =
_lifetimeScope.ResolveOptional<IValida
tor<TCommand>>();

            if (validator == null)
            {
                throw new
MissingCommandValidatorException<T
Command>(command);
            }

            var validationResult = await
validator.ValidateAsync(command);

            if (validationResult == null)
            {
                throw new
CorruptedValidatorException<IValidator
<TCommand>, TCommand>(validator,
command);
            }

            if (!validationResult.IsValid)
            {
                throw new
ValidationException(command,
validationResult.Errors);
            }
        }
    }
}
using System;
using System.Threading.Tasks;
using Autofac;
using FluentValidation;
using SchoolSystem.Abstractions.CQRS;
using
SchoolSystem.Application.Exceptions.Q
ueries;
using
SchoolSystem.Application.Exceptions.V
alidation;
using ValidationException =
SchoolSystem.Application.Exceptions.V
alidation.ValidationException;

namespace
SchoolSystem.Application.Buses
```

```
{
    //TODO: add module

    internal class QueryBus : IQueryBus
    {
        private readonly ILifetimeScope
_lifetimeScope;

        public QueryBus(ILifetimeScope
lifetimeScope)
        {
            _lifetimeScope = lifetimeScope;
        }

        public async Task<TQueryResult>
Execute<TQuery,
TQueryResult>(TQuery query)
            where TQuery :
IQuery<TQueryResult>
            where TQueryResult :
IQueryResult
        {
            var queryHandler =
ResolveHandler<TQuery,
TQueryResult>(query);

            await Validate<TQuery,
TQueryResult>(query);

            return await
queryHandler.Execute(query);
        }

        private IQueryHandler<TQuery,
TQueryResult>
ResolveHandler<TQuery,
TQueryResult>(TQuery query)
            where TQuery :
IQuery<TQueryResult>
            where TQueryResult :
IQueryResult
        {
            var queryHandler =
_lifetimeScope.ResolveOptional<IQuery
Handler<TQuery, TQueryResult>>();

            if (queryHandler == null)
            {
                throw new
QueryHandlerNotFoundException<TQu
ery, TQueryResult>(query);
            }

            return queryHandler;
        }

        private async Task
Validate<TQuery,
TQueryResult>(TQuery query)
            where TQuery :
IQuery<TQueryResult>
            where TQueryResult :
IQueryResult
        {
            var validator =
_lifetimeScope.ResolveOptional<IValida
tor<TQuery>>();

            if (validator == null)
            {
                throw new
MissingQueryValidatorException<TQue
ry, TQueryResult>(query);
            }

            var validationResult = await
validator.ValidateAsync(query);

            if (validationResult == null)
            {
                throw new
CorruptedValidatorException<IValidator
<TQuery>, TQuery>(validator, query);
            }
```

```
            if (!validationResult.IsValid)
            {
                throw new
ValidationException(query,
validationResult.Errors);
            }
        }
    }
}
namespace
SchoolSystem.Abstractions.CQRS
{
    public interface ICommand
    {
    }
}
using System.Threading.Tasks;

namespace
SchoolSystem.Abstractions.CQRS
{
    public interface ICommandBus
    {
        Task
Execute<TCommand>(TCommand
command)
            where TCommand : ICommand;
    }
}
using System.Threading.Tasks;

namespace
SchoolSystem.Abstractions.CQRS
{
    public interface
ICommandHandler<TCommand>
        where TCommand : ICommand
    {
        Task Execute(TCommand
command);
    }
}
namespace
SchoolSystem.Abstractions.CQRS
{
    public interface IQuery<TResult>
where TResult : IQueryResult
    {
    }
}
using System.Threading.Tasks;

namespace
SchoolSystem.Abstractions.CQRS
{
    public interface IQueryBus
    {
        Task<TQueryResult>
Execute<TQuery,
TQueryResult>(TQuery query)
            where TQuery :
IQuery<TQueryResult>
            where TQueryResult :
IQueryResult;
    }
}
using System.Threading.Tasks;

namespace
SchoolSystem.Abstractions.CQRS
{
    public interface
IQueryHandler<TQuery, TQueryResult>
        where TQuery :
IQuery<TQueryResult>
        where TQueryResult : IQueryResult
    {
        Task<TQueryResult>
Execute(TQuery query);
    }
}
namespace
SchoolSystem.Abstractions.CQRS
```

```
{
    public interface IQueryResult
    {
    }
}

import React, { Component } from
"react";
import AddSchool from "./add-school-
component";

class AddSchoolContainer extends
Component {
  constructor(props) {
    super(props);

    this.state = {
      school: {},
      cities: []
    };

    this.onSubmit =
this.onSubmit.bind(this);
    this.onChange =
this.onChange.bind(this);
    this.getCities =
this.getCities.bind(this);
    this.getCitiesFromLocalStorage =
this.getCitiesFromLocalStorage.bind(this
);
    this.saveSchool =
this.saveSchool.bind(this);
    this.validateSchool =
this.validateSchool.bind(this);
    this.handleValidationFailures =
this.handleValidationFailures.bind(this);
  }

  componentDidMount() {
    let getCitiesResult = this.getCities();
    getCitiesResult.then(cities => {
      this.setState({ cities });
    });
  }

  async getCities() {
    let cities =
this.getCitiesFromLocalStorage();

    if (!cities || cities.length <= 0) {
      cities = await this.fetchCities();

window.localStorage.setItem("cities",
JSON.stringify(cities));
    }
    return cities;
  }

  getCitiesFromLocalStorage() {
    let cities =
window.localStorage.getItem("cities");
    return JSON.parse(cities);
  }

  async fetchCities() {
    return
fetch("http://localhost:8001/api/cities", {
      method: "GET",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json"
      }
    })
      .then(response => response.json())
      .then(response => {
        let mappedCities =
this.mapCities(response.cities);
        return mappedCities;
      });
  }

  mapCities(cities) {
    if (!(cities && cities.length > 0)) {
      return [];
```

```
    }
    return cities.map(function(city) {
      let mappedCity = {
        id: city.id,
        value: city.id,
        displayValue: city.name
      };

      return mappedCity;
    });
  }

  onSubmit(event) {
    event.preventDefault();
    this.validateSchool(this.state.school)
&&
      this.saveSchool(this.state.school);
  }

  validateSchool(school) {
    let errors = [];

    school.password !==
school.repeatPassword &&
      errors.push({ name: "password",
message: "passwords doesn't match" });

    if (errors && errors.length > 0) {
      this.setState({ errors: errors });
      return false;
    }
    return true;
  }

  saveSchool(school) {

fetch("http://localhost:8001/api/Schools",
{
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json"
      },
      body: JSON.stringify(school)
    })
      .then(response => {
        return response.status !== 200 &&
response.json();
      })
      .then(response => {
        if (!response) {
          this.props.history.push("/");
        }
        if (
          response.Content &&
          response.Content.Code &&
          response.Content.Code === "cqrs-
0008"
        ) {

this.handleValidationFailures(response.C
ontent.ValidationFailures);
        }
      })
      .catch(reason =>
console.error(reason));
  }

  handleValidationFailures(failures) {
    let mappedFailures =
failures.map((failure, index) => {
      let mappedFailure = {
        key: index,
        name: failure.PropertyName,
        message: failure.ErrorMessage
      };
      return mappedFailure;
    });

    this.setState({ errors: mappedFailures
});
  }

  onChange(event) {
```

```
    let school = this.state.school;
    school[event.target.name] =
event.target.value;
    this.setState({ school });
  }

  render() {
    return (
      <AddSchool
        errors={this.state.errors}
        school={this.state.school}
        cities={this.state.cities}
        onChange={this.onChange}
        onSubmit={this.onSubmit}
      />
    );
  }
}

export default AddSchoolContainer;

import React from "react";
import Form from
"../common/form/form-component";
import FormFieldTypes from
"./../common/form/form-field-types";
import { Link } from "react-router-dom";

const AddSchool = props => {
  function getFields() {
    return [
      {
        type: FormFieldTypes.INPUT,
        inputType: "text",
        labelContent: "Name",
        name: "name",
        id: "name-input",
        value: props.school.name,
        onChange: props.onChange
      },
      {
        type: FormFieldTypes.INPUT,
        inputType: "text",
        labelContent: "Login",
        name: "login",
        id: "login-input",
        value: props.school.login,
        onChange: props.onChange
      },
      {
        type: FormFieldTypes.INPUT,
        inputType: "password",
        labelContent: "Password",
        name: "password",
        id: "password-input",
        value: props.school.password,
        onChange: props.onChange
      },
      {
        type: FormFieldTypes.INPUT,
        inputType: "password",
        labelContent: "Repeat password",
        name: "repeatPassword",
        id: "repeat-password-input",
        value: props.school.repeatPassword,
        onChange: props.onChange
      },
      {
        id: "cities-select",
        type: FormFieldTypes.SELECT,
        labelContent: "City",
        name: "cityId",
        options: props.cities,
        onChange: props.onChange
      }
    ];
  }

  return (
    <div className="centralized-
container">
      <Link to="/" className="link-
button">
        back
```

```
    </Link>
    <Form
      class="add-school-form"
      submitButtinText="Add"
      formHeaderText="Add school"
      fields={getFields()}
      onSubmit={props.onSubmit}
      errors={props.errors}
    />
   </div>
  );
};

export default AddSchool;

import React from "react";
import "./App.css";
import SchoolsListContainer from
"../schools-list/schools-list-component-
container";
import EditSchoolContainer from
"../edit-school/edit-school-component-
container";
import AddSchoolContainer from
"../add-school/add-school-component-
container";
import { Route } from "react-router-
dom";

function App() {
 return (
  <React.Fragment>
   <Route path="/" exact
component={SchoolsListContainer} />
   <Route path="/edit/:id"
component={EditSchoolContainer} />
   <Route path="/add"
component={AddSchoolContainer} />
  </React.Fragment>
 );
}

export default App;

import React, { Component } from
"react";
import FormFieldTypes from "./form-
field-types";
import "./form-component.css";

class Form extends Component {
 constructor(props) {
  super(props);

  this.parseField =
this.parseField.bind(this);
 }

 parseField(field) {
  switch (field.type) {
   case FormFieldTypes.SELECT: {
    return <FormSelect key={field.id}
{...field} />;
   }
   case FormFieldTypes.INPUT: {
    return <FormInput key={field.id}
{...field} />;
   }
   default: {
    console.log("render default item");
   }
  }
 }

 render() {
  let fields = this.props.fields.map(field
=> this.parseField(field));

  return (
   fields &&
   fields.length > 0 && (
    <div className={`form-container
${this.props.class}`}>
     {this.props.formHeaderText && (
```

```
          <h3
className="form__header">{this.props.
formHeaderText}</h3>
        )}
        <form
onSubmit={this.props.onSubmit}
className="form">
          {fields}
          <button
            className={`submit-button
${this.props.submitButtonClassName}`}
            type="submit"
          >
            {this.props.submitButtonText}
          </button>
        </form>
        {this.props.errors &&
<FormErrorsBlock
errors={this.props.errors} />}
      </div>
    )
  );
 }
}

export default Form;

Form.defaultProps = {
 fields: [],
 submitButtonClassName: "",
 submitButtonText: "Submit"
};

const FormItem = props => (
 <div className="input-container">
  <label
htmlFor={props.name}>{props.labelCon
tent}</label>
  {props.children}
 </div>
);

const FormInput = props => (
 <FormItem {...props}>
  <input
   type={props.inputType || "text"}
   name={props.name}
   id={props.id}
   onChange={props.onChange}
   value={props.value}
   required={props.required && true}
  />
 </FormItem>
);

const FormSelect = props => {
 return (
  <FormItem {...props}>
   <select
    name={props.name}
    onChange={props.onChange}
    required={props.required && true}
   >
    <option value="">None</option>
    {props.options &&
     props.options.map(option => (
      <option key={option.id}
value={option.value}>
       {option.displayValue}
      </option>
     ))}
   </select>
  </FormItem>
 );
};

const FormErrorsBlock = props => {
 return (
  <div className="form-
container_errors_container">
   <ul>
    {props.errors.map(error => (
     <li
key={error.key}>{error.message}</li>
```

```
      ))}
    </ul>
   </div>
  );
};

const FormFieldTypes = {
  INPUT: "input",
  SELECT: "select"
};

export default FormFieldTypes;

import React from "react";
import "./message-component.css";

const Message = props => {
  return (
    <div className="message-block">
      <p className="message-
block__message">{props.message}</p>
    </div>
  );
};

export default Message;

import React, { Component } from
"react";
import EditSchool from "./edit-school-
component";

class EditSchoolContainer extends
Component {
  constructor(props) {
    super(props);

    this.state = {
      school: {},
      cities: []
    };

    this.onChange =
this.onChange.bind(this);
    this.updateSchool =
this.updateSchool.bind(this);
    this.onSubmit =
this.onSubmit.bind(this);
    this.loadCities =
this.loadCities.bind(this);
    this.loadSchool =
this.loadSchool.bind(this);
    this.mapCities =
this.mapCities.bind(this);
    this.handleValidationFailures =
this.handleValidationFailures.bind(this);
  }

  componentDidMount() {
    this.loadSchool();
    this.loadCities();
  }

  loadCities() {
    fetch("http://localhost:8001/api/cities",
{
      method: "GET",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json"
      }
    })
      .then(response => response.json())
      .then(response => {
        this.setState({ cities:
this.mapCities(response.cities) });
      });
  }

  mapCities(cities) {
    if (!(cities && cities.length > 0)) {
      return [];
    }
    return cities.map(function(city) {
```

```
      let mappedCity = {
        id: city.id,
        value: city.id,
        displayValue: city.name
      };

      return mappedCity;
    });
  }

  loadSchool() {
    let storage = window.localStorage;
    let schools =
JSON.parse(storage.getItem("schools"));
    let currentSchool = schools.find(
      school => school.id ===
this.props.match.params.id
    );

    this.setState({ school: currentSchool
});
  }

  onChange(e) {
    let currentSchool = this.state.school;
    currentSchool[e.target.name] =
e.target.value;
    this.setState({ school: currentSchool
});
  }

  onSubmit(e) {
    e.preventDefault();
    this.updateSchool(this.state.school);
  }

  updateSchool(school) {

fetch("http://localhost:8001/api/Schools",
{
      method: "PUT",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json"
      },
      body: JSON.stringify(school)
    })
      .then(response => {
        return response.status !== 200 &&
response.json();
      })
      .then(response => {
        if (!response) {
          this.props.history.push("/");
        }
        if (
          response.Content &&
          response.Content.Code &&
          response.Content.Code === "cqrs-
0008"
        ) {

this.handleValidationFailures(response.C
ontent.ValidationFailures);
        }
      })
      .catch(reason =>
console.error(reason));
  }

  handleValidationFailures(failures) {
    let mappedFailures =
failures.map((failure, index) => {
      let mappedFailure = {
        key: index,
        name: failure.PropertyName,
        message: failure.ErrorMessage
      };
      return mappedFailure;
    });

    this.setState({ errors: mappedFailures
});
  }
```

```
  render() {
    return (
      <EditSchool
        errors={this.state.errors}
        school={this.state.school}
        cities={this.state.cities}
        onChange={this.onChange}
        onSubmit={this.onSubmit}
      />
    );
  }
}

export default EditSchoolContainer;

import React from "react";
import { Link } from "react-router-dom";
import Form from
"../common/form/form-component";
import FormFieldTypes from
"./../common/form/form-field-types";

const EditSchool = props => {
  function getFields() {
    return [
      {
        type: FormFieldTypes.INPUT,
        inputType: "text",
        labelContent: "Name",
        name: "name",
        id: "name-input",
        value: props.school.name,
        onChange: props.onChange,
        required: false
      },
      {
        id: "cities-select",
        type: FormFieldTypes.SELECT,
        labelContent: "City",
        name: "cityId",
        options: props.cities,
        onChange: props.onChange,
        required: false
      }
    ];
  }

  return (
    <div className="centralized-
container">
      <Link to="/" className="link-
button">
        back
      </Link>
      <Form
        class="edit-school-form"
        formHeaderText="Edit school"
        submitButtinText="Edit"
        fields={getFields()}
        onSubmit={props.onSubmit}
        errors={props.errors}
      />
    </div>
  );
};

export default EditSchool;

import React, { Component } from
"react";
import SchoolsList from "./schools-list-
component";

class SchoolsListContainer extends
Component {
  constructor(props) {
    super(props);
    this.state = {
      schools: []
    };

    this.fetchSchools =
this.fetchSchools.bind(this);
```

```
    this.deleteSchool =
this.deleteSchool.bind(this);
  }

  componentDidMount() {
    this.fetchSchools();
  }

  fetchSchools() {
    return
fetch("http://localhost:8001/api/Schools",
{
      method: "GET",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json"
      }
    })
      .then(response => response.json())
      .then(response => {
        this.setState({ schools:
response.schools });
        window.localStorage.setItem(
          "schools",
          JSON.stringify(response.schools)
        );
      })
      .catch(reason => {
        console.error(reason);
      });
  }

  deleteSchool(id) {

fetch(`http://localhost:8001/api/Schools/
${id}`, {
      method: "DELETE",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json"
      }
    })
      .then(() => {
        let schools =
this.state.schools.slice();
        schools = schools.filter(school =>
school.id !== id);

window.localStorage.setItem("schools",
JSON.stringify(schools));
        this.setState({ schools });
      })
      .catch(reason =>
console.error(reason));
  }

  render() {
    return (
      <SchoolsList
        schools={this.state.schools}
        deleteSchool={this.deleteSchool}
      />
    );
  }
}

export default SchoolsListContainer;

import React, { Component } from
"react";
import "./schools-list-component.css";
import { Link } from "react-router-dom";
import Message from
"./../common/message/message-
component";

const SchoolsList = props => {
  function getTable() {
    return props.schools &&
props.schools.length > 0
      ? renderTable()
      : renderNoContent();
  }
```

```
  function renderTable() {
    return (
      <table className="schools-list">
        <caption className="schools-
list__header">Schools</caption>
        <thead>
          <SchoolsListHeader />
        </thead>
        <tbody>{renderSchools()}</tbody>
      </table>
    );
  }

  function renderSchools() {
    return props.schools.map(school => {
      return (
        <SchoolsListItem
          name={school.name}
          city={school.city}
          login={school.login}
          password={school.password}
          id={school.id}
          key={school.id}
          deleteSchool={() =>
props.deleteSchool(school.id)}
        />
      );
    });
  }

  function renderNoContent() {
    return <Message message="Schools
list is empty" />;
  }

  return (
    <div className="centralized-
container">
      <Link to="/add" className="link-
button">
        Add new
      </Link>
      {getTable()}
    </div>
  );
};

const SchoolsListHeader = props => (
  <tr>
    <th>Name</th>
    <th>City</th>
    <th>Login</th>
    <th>Actions</th>
    <th />
  </tr>
);

class SchoolsListItem extends
Component {
  constructor(props) {
    super(props);

    this.showPassword =
this.showPassword.bind(this);
  }

  showPassword(e) {
    let passwordInput =
document.getElementById("password_"
+ this.props.id);
    passwordInput.type =
passwordInput.type === "text" ?
"password" : "text";
  }

  render() {
    return (
      <tr>
        <td>{this.props.name}</td>
        <td>{this.props.city}</td>
        <td>{this.props.login}</td>
        {/* <td>
          <input
            type="password"
```

```
            name="password"
            id={"password_" + this.props.id}
            value={this.props.password}
            disabled
          />
        </td> */}
        <td className="schools-
list__action-buttons-container">
          {/* <button
            className="schools-
list__action-button"
            onClick={this.showPassword}
          >
            show password
          </button> */}
          <Link to={"/edit/" + this.props.id}
className="link-button">
            edit
          </Link>
          <button
            className="schools-
list__action-button"

onClick={this.props.deleteSchool}
          >
            delete
          </button>
        </td>
      </tr>
    );
  }
}

export default SchoolsList;

import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from
"./components/app/App";
import { BrowserRouter as Router }
from "react-router-dom";

ReactDOM.render(
  <Router>
    <App />
  </Router>,
  document.getElementById("root")
);
```